Universidad Autónoma de Querétaro

Facultad de ingeniería.

Sensor Inteligente basado en FPGA para la cuantificación de distribuciones espaciales entre objetos.

Tesis.

Que como parte de los requisitos para obtener el grado de Maestro en Ciencias (Mecatrónica).

Presenta.

Juan Angel Ramírez Núñez.

Dirigido por:

Dr. Roque Alfredo Osornio Rios.

Santiago de Querétaro, Qro, México, Noviembre 2014.

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias Mecatrónica

NOMBRE DE LA TESIS
Sensor Inteligente basado en FPGA para la cuantificación de distribuciones espaciales entre objetos.

TESIS
Que como parte de los requisitos para obtener el grado de
Maestro en Ciencias Mecatrónica

Presenta:
Juan Angel Ramírez Núñez

Dirigido por:
Roque Alfredo Osornio Rios

SINODALES

Dr. Roque Alfredo Osornio Rios Presidente	Firma
Dr. Luis Alberto Morales Hernández Secretario	Firma
Dr. Juan Primo Benítez Rangel Vocal	Firma
Dr. J. Jesús de Santiago Pérez Suplente	Firma
Dr. Iván R. Terol Villalobos Suplente	Firma

Dr. Aurelio Domínguez González
Director de la Facultad

Dr. Irineo Torres Pacheco
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Mayo 2014
México

Resumen.

En las fundiciones nodulares determinar los tamaños de los nódulos (espectro de tamaños) a partir de imágenes metalográficas es una tarea que comúnmente se realiza con sistemas por software, sin en cambio, esto conlleva a tener una PC destinada solamente para la realización de esta tarea, y requiere de una considerable carga computacional, además de que el tiempo de ejecución es alto. En este trabajo se realizó la descripción digital de la rutina de un algoritmo, que permite calcular el espectro de tamaños e implementarla en hardware con ayuda de un FPGA (Altera DE2-115), para minimizar la carga computacional y disminuir el tiempo de ejecución de manera considerable; obteniendo a su vez un sistema dedicado a realizar esta tarea de manera eficiente e independiente de una PC. Se utilizaron los operadores morfológicos erosión y dilatación para calcular el espectro de tamaños, se realizó la descripción digital de éstos para implementar el algoritmo en hardware, el kernel o elemento estructural utilizado es una máscara de 3x3, una vez calculados los espectros de tamaños de la imagen se determina el tamaño máximo de los nódulos; se realizaron las pruebas en imágenes metalográficas en escala de grises con un tamaño de 656x494 pixeles, con lo que se logró reducir la carga computacional y el tiempo de procesamiento de una manera considerable comparado con el cálculo en software; además otro punto a destacar es que al ser un diseño digital de nuestra propiedad, se puede acoplar a otros módulos y así poder crear en trabajos futuros un sensor inteligente que permita obtener el tamaño de los nódulos sin la necesidad de hacer uso de un PC para ello.

(**Palabras clave:** FPGA, operadores morfológicos, sensor inteligente)

Abstract.

In nodular foundries, determining the nodular sizes (the size spectrum) of metallographic images is a work that it´s realized by software systems, although, this work uses a PC only for this task, and this routine requires a considerable computing load and a high execution time. In this work digital description of algorithm is realized, permitting to calculate the spectrum sizes and it is implemented in hardware based on an FPGA (Altera DE2-115), to minimize the computing load and decrease the execution time in a considerable way; obtaining a dedicated system to make this task efficiently and independent of a PC. The erosion and dilatation morphological operators have been used to calculate the sizes spectrum, digital description of these was made to implement the algorithm in hardware, the kernel used is a 3x3 mask, with the image sizes spectrum the maximum nodular size of the image can be determined, metallographic images testing was realized in gray scale with size of 656x494 pixels, with this, the computing load is reduced and also the processing time in comparison whit software implementations; in addition, the I.P. Cores are ours and thanks to this is possible create a smart sensor that will be used to obtain the nodule size without PC.

(**Word keys:** FPGA, morphological operators, smart sensor)

Agradecimientos.

Doy gracias a dios, a mi familia y amigos por el gran apoyo que siempre me han brindado en las metas y proyectos que me propongo realizar.

Agradezco a todos mis amigos del plantel por haberme brindado su amistad y confianza a lo largo de estos dos años que duro la maestría.

Un agradecimiento muy grande y especial a mis asesores de tesis el Dr. Roque Alfredo Osornio Rios y Dr. Luis Alberto Morales Hernández por haber tenido mucha paciencia hacia mí y haber estado ahí para aclarar todas las dudas que surgían al estar realizando el proyecto.

A los integrantes del grupo HSP Digital por su apoyo y consejos al momento de realizar mis descripciones digitales en especial al Dr. Luis Morales Velázquez.

A CONACYT por la beca proporcionada durante estos dos años y a la Universidad Autónoma de Querétaro.

Índice

ÍNDICE DE FIGURAS

Figura **Página**

Figura **Página**

Figura **Página**

Figura **Página**

Índice de Tablas.

Tabla **Página**

Capítulo 1.
Introducción

El estudio de las características micro-estructurales de un material permite predecir el comportamiento de éste por la relación que existe entre dichas características y las propiedades mecánicas del material. En el caso de metales y aleaciones, la metalografía es la disciplina que permite realizar la caracterización de una muestra representativa del material mediante el uso de microscopios para comparar y cuantificar los patrones micro-estructurales, relaciones espaciales y de forma, con la finalidad de obtener datos numéricos. Aun cuando las características mencionadas anteriormente son fácilmente reconocibles en una imagen, la obtención de datos numéricos confiables de forma manual es muy ineficiente debido a que depende de un evaluador. Obtener un dato que de la relación espacial entre objetos de manera automática permite que el dato obtenido sea confiable.

Los desarrollos en hardware y software computacionales de los últimos años han hecho posible un cambio significativo en los métodos de caracterización de materiales, permitiendo la extracción cuantitativa de información de forma automática mediante la adquisición, procesamiento y análisis de imágenes en forma digital (ASM Handbook,2004), ya sea con la digitalización de una fotografía metalográfica o con la adquisición directa de una cámara en el microscopio para su posterior procesamiento y análisis en una PC o en una plataforma especializada, eliminado de esta manera los errores que pueden ser introducidos por el evaluador. Una de las técnicas de procesamiento de imágenes usadas en el análisis metalográfico es la morfología matemática, la cual se basa en el estudio de la forma y se utiliza para investigar la relación que existe entre una imagen y un elemento estructural.

Esta tesis esta seccionado en 5 capítulos, en el primer capítulo se hace una descripción de la idea general de la misma, en éste también se plantean los objetivos, la hipótesis y la justificación del trabajo; en el segundo capítulo se presenta la fundamentación teórica, es decir, los temas y herramientas necesarias para la solución del problema; en el tercer capítulo se muestra la metodología o pasos a seguir para llevar a cabo la tesis; en el capítulo 4 se muestran los resultados obtenidos y por último en el capítulo 5 se describen las conclusiones y perspectivas.

1.1. Antecedentes

En la Universidad Autónoma de Querétaro (UAQ) se han desarrollado trabajos sobre el procesamiento digital de imágenes, debido a que es una de las líneas de investigación importante que interactúan con otras áreas y por ende con la mecatrónica, por ejemplo, Vargas (2000) logró la detección de cisuras del cerebro mediante filtros morfológicos, estos implementados en software. Méndez (2008) implementó técnicas de procesamiento digital de imágenes logrando con ello la detección automática de unidades formadoras de colonias. Morales (2005) propuso la caracterización de huellas digitales a partir de una familia de filtros morfológicos direccionales.

Todos los trabajos fueron implementados en software a través de una PC (*Personal Computer, computadora personal*) para el procesamiento de los algoritmos; sin embargo, los FPGA (*Field Programmable Gate Array*, Arreglo de Compuertas Programables en Campo) han mostrado su capacidad para procesamiento en diversas aplicaciones industriales y también en lo relacionado al procesamiento de imágenes, fue así que Ramos (2010) implementó la metodología de una etapa básica de un sistema de procesamiento de imágenes basado en FPGA.

A nivel nacional se han desarrollado plataformas en hardware sobre el procesamiento digital de imágenes, por ejemplo, Córdova (2010) llevó a cabo el diseño de un módulo en FPGA para realizar el ajuste de contraste de una imagen a través de la técnica de igualación de histograma. Rangel (2005) realizó la implementación en FPGA de algoritmos de morfología matemática para el procesamiento de imágenes binarias, así también la implementación de controladores para la adquisición y binarización de imágenes directamente de una cámara y el desplegado de imágenes en un monitor VGA, con lo que se creó un sistema de procesamiento morfológico de imágenes y que se puede operar sin el uso de una PC, con el inconveniente de que no puede ser programado, por lo que cualquier cambio requiere configuración. Razo (2006) desarrolló un procesador morfológico de imágenes basado en FPGA que permitió la adquisición de las imágenes a través de una cámara digital OV7648, el desplegado de imágenes en monitor VGA, seleccionar el tipo de filtro a utilizar y seleccionar el nivel de procesamiento morfológico, aumentado la versatilidad del sistema, pero con el inconveniente de requerir el uso de una PC para él envió de estos comandos, además de estar restringida a un sistema de desarrollo y a una cámara en particular. González (2012) desarrolló un procesador morfológico de imágenes en FPGA aplicado al análisis metalográfico y gracias a los módulos desarrollados se logró reducir los tiempos de procesamiento, obteniendo así un sistema capaz de

adquirir, procesar y desplegar imágenes directamente, pero solo cuenta con los operadores morfológicos básicos.

En el ámbito internacional, Shao y Liang (2008) propusieron la arquitectura de un acelerador en hardware para la segmentación de objetos en videos, a través del desarrollo morfológico de imágenes reconfigurable, donde se propone un set de instrucciones que controlan la operación y conectividad de una serie de elementos de procesamiento que en conjunto conforman este procesador.

Los sensores inteligentes están definidos por el estándar IEEE 1451 como sensores con memoria pequeña y la conexión física estandarizada para permitir la comunicación con el procesador y una red de datos. Más allá de esta definición, un sensor inteligente se define como la combinación de un sensor con acondicionamiento de señales, algoritmos embebidos e interfaz digital (Gervais, 2011).

Con respecto a trabajos relacionados a sensores inteligentes, Contreras-Medina et al. (2012) desarrollaron un sensor inteligente basado en FPGA, para la cuantificación de síntomas de plantas enfermas logrando así identificar varias enfermedades comunes en éstas. Trejo et al. (2010) desarrollaron un sensor inteligente obteniendo tres veces mejor precisión en la medición del desgaste de la herramienta de una máquina CNC. Vera et al. (2011) generaron un sensor inteligente para medir desplazamientos en línea en aplicaciones de alta velocidad usando interferómetros heterodinos. Rodríguez et al. (2010) desarrollaron una red de sensor inteligente para calcular la cinemática directa de un robot industrial y con ello mejorar el rendimiento del robot.

De acuerdo a los antecedentes es evidente que se han desarrollado diversos trabajos tanto de sensores inteligentes como aquellos que utilizan morfología matemática, sin embargo, la mayoría de ellos están implementados en software; aunque ya hay trabajos de procesamiento en hardware específicamente en *FPGA* ninguno de ellos aborda el problema de las distribuciones espaciales, lo que destaca que a la fecha no se ha generado un sensor inteligente en hardware para la cuantificación de distribuciones espaciales.

Es por ello que en esta investigación se propone desarrollar un sensor inteligente basado en FPGA, para realizar el análisis mediante un procesador morfológico para obtener la distribución espacial entre objetos, utilizando las técnicas de dilatación, erosionado, apertura y cerradura, y así obtener dicho resultado en el menor tiempo y usando el mínimo de recursos computacionales posibles.

1.2 Objetivos e Hipótesis

1.2.1 Objetivo general

- Integrar un sensor inteligente basado en FPGA y en una cámara comercial para la determinación de un índice que cuantifique la distribución espacial de los objetos.

1.2.2 Objetivos particulares

- Determinar los tamaños particulares de los objetos mediante operadores morfológicos que realicen una cuantificación espacial de los mismos.

- Establecer un índice de distribución espacial mediante la determinación de un criterio basado en el cociente entre distancias para la medición cuantitativa de dicha distribución.

- Desarrollar un procesador basado en FPGA que ejecute los operadores morfológicos necesarios para establecer un índice de distribución espacial.

- Integrar un sensor inteligente basado en una cámara comercial y un procesador morfológico basado en FPGA que permita hacer mediciones cuantitativas de distribución espacial de objetos.

1.2.3 Hipótesis

- La morfología matemática y el procesamiento basado en FPGA permitirá determinar un índice para cuantificar la distribución espacial entre objetos, mejorando así el tiempo de procesamiento y los recursos computacionales respecto a los métodos hasta ahora utilizados.

1.3 Justificación

Realizar el procesamiento de imágenes por medio de software es inviable por el tiempo de ejecución del algoritmo, mucho mayor que los realizados en hardware (por ejemplo en un FPGA).

No depender de dispositivos electrónicos para llevar a cabo la captura de datos, en este caso imágenes, involucra que se podrá usar una gran variedad de equipos de diferentes marcas; así el usuario podrá adquirir el dispositivo de captura de acuerdo a su presupuesto y no a uno en particular.

Realizar la descripción de algoritmos en hardware, implica que el costo para llevar a cabo esto, es mucho menor que si se implementará en software.

Utilizar un sensor inteligente permite llevar a cabo todo un proceso, por ende, obtener un resultado de una manera independiente.

1.4 Planteamiento General

Actualmente existe una gran variedad de soluciones comerciales para llevar a cabo el procesamiento digital de imágenes. Estas pueden ser tanto en software como en hardware, aunque sí de menor tiempo de ejecución se trata es conveniente realizar dicho proceso en una plataforma completamente basada en hardware.

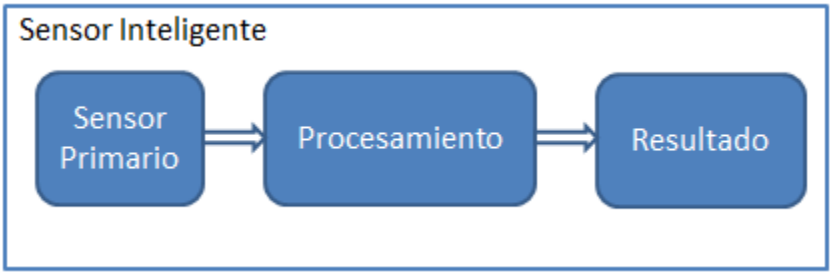
El tiempo necesario para llevar a cabo el procesamiento digital de una imagen por medio de software conlleva que se utilice una gran cantidad de tiempo en su procesamiento, así como la necesidad de usar una PC. En contraste, existen aplicaciones donde es necesario que dicho análisis se ejecute en el menor tiempo posible y utilizando menos recursos computacionales.

La dependencia de dispositivos electrónicos para llevar a cabo un proceso en particular, es ineficiente al no poder sustituir dichos dispositivos con otros de diferente proveedor.

Con respecto a plataformas comerciales que integren los elementos de software y equipos de hardware para el procesamiento digital de imágenes, existe la empresa LabView, que permite el desarrollo de este tipo de aplicaciones de una manera sencilla utilizando bloques, pero con la desventaja de que los equipos son de alto costo alto costo (cámaras arriba de $20,000.00, módulos en hardware para la adquisición de imagen por varios protocolos(USB, Ethernet, etc.) arriba de $4,000.00, software NI Vision Develpoment $54 395.00), de configuración establecida y mayor uso de tiempo de ejecución y recursos computacionales.

Por ello, una plataforma completamente basada en hardware para el procesamiento digital de imágenes a través de un procesador morfológico permitirá que el tiempo de ejecución de los mismos sea realizado en un menor tiempo y costos con respecto a los desarrollados en software.

A continuación se muestra un diagrama a bloques de manera muy general de cómo se pretende dar solución a este trabajo de tesis:



1.4.1 Diagrama de bloques general para dar solución al problema.

Una vez que se tiene la imagen adquirida por el sensor primario los dato de la misma son enviados a ser procesados a el FPGA utilizando algún protocolo de comunicación (RS232, Ethernet, USB, etc), una vez calculado los datos de interés los resultados son enviados nuevamente ya sea a un monitor, un LCD, un PC, etc.

Capítulo 2.
Fundamentación Teórica

En este capítulo se presenta la evolución en el área del procesamiento digital de imágenes a lo largo de la historia, así también se explican las herramientas (algoritmos) y plataforma que son utilizados para el desarrollo de esta tesis.

2.1 Estado del arte

Uno de las primeras aplicaciones de las imágenes digitales fue en el sector periodístico, cuando las imágenes fueron enviadas por primera vez por medio de un cable submarino entre Londres y New York. Gracias al sistema Bartlane, llamado así por sus inventores Harry G. Bartholomew and Maynard D. McFarlane, en los años 1920's se redujo el tiempo de transportar una imagen a través del océano Atlántico de más de una semana a menos de tres horas, por ende, se crearon equipos especializados en reconstruir y trasmitir estas imágenes o información. Las imágenes fueron inicialmente codificadas a 5 niveles de grises y ya para 1929 este número incremento a 15.

Pero fue hasta los años 1960's que se utilizó una computadora lo suficientemente capaz de realizar la tarea de hacer algún procesamiento a una imagen; y fue así que en 1964 se realizó la primer restauración de una imagen tomada de la superficie de la luna.

También entre los años 1960´s y 1970´s se comenzó a hacer uso del procesamiento digital de imágenes en el área de medicina, observaciones remotas a la tierra con ayuda de satélites y en astronomía.

A principio de los años 70's se inventó la tomografía axial computarizada (CAT, C*omputerized Axial Ttomography)* también llamada tomografía computarizada (CT) siendo el evento más importantes de la aplicación el procesamiento digital de imágenes. El algoritmo utilizado para realizar la tomografía fue diseñado por Sir Godfrey N. Hounsfield y el profesor Allan M. Cormack, quienes gracias a su invento ganaron el premio nobel en medicina en 1979 (Nobel Prize in Physiology or Medicine, 1979).

Actualmente el procesamiento digital de imágenes tiene una gran cantidad de aplicaciones en muchas áreas y también gracias al desarrollo computacional se

han desarrollado una gran cantidad de algoritmos que ayudan a realizar mejor dichos procesos.

2.2 Introducción al procesamiento de imágenes

Una imagen puede ser representada como una función $f(x,y)$, con (x,y) las coordenadas de los pixeles (elementos de la imagen) dentro la imagen. La función de salida de la imagen es el valor del pixel de la imagen, el cual es un valor lógico de 0 o 1 para imágenes binarias, mientras que para imágenes en niveles de gris la función tiene valores entre 0 a 255 (0,…255). En procesamiento de imágenes se modifica la imagen de entrada por medio de un proceso el cual es diseñado para cumplir una tarea específica, por ejemplo, remover el ruido en una imagen para obtener a la salida una imagen libre de ruido. El procesamiento de una imagen es realizado entonces por una transformación ψ de una imagen de entrada $f(x,y)$ a una imagen de salida $g(x,y)$. Se puede obtener la ecuación entrada-salida como se indica en la ecuación 2.1(Goutsias y Batman, 2000; Sivakumar y Goutsias,1997):

$$g(x,y)= \psi(x,y) \quad (2.1)$$

La selección del procesamiento de la imagen puede reducirse a asumir dos propiedades fundamentales:

Lineal (aditividad): el operador ψ produce el mismo resultado cuando se le aplica a la suma de dos imágenes, o cuando se aplica de manera separada sobre cada una de ella sumando las imágenes de salida.

Invariante a la traslación: el operador ψ produce el mismo resultado cuando la translación es aplicada a la imagen o al resultado de la operación.

El operador ψ es llamado lineal cuando este satisface las siguientes propiedades:

$$\psi\big(f_1(x,y) + f_2(x,y)\big) = \psi\big(f_1(x,y)\big) + \psi\big(f_2(x,y)\big) \quad (2.2)$$

$$\psi\big(cf(x,y)\big) = c\psi\big(f(x,y)\big) \quad (2.3)$$

Donde c es una translación (valor constante). La ecuación 2.2 es la propiedad de aditividad para el filtro lineal. Esta plantea que la suma de dos imágenes puede ser procesada por el operador o que el resultado del procesamiento de estas dos imágenes puede ser sumado y que ambos criterios regresan la misma imagen de salida.

Estrictamente hablando, los operadores lineales no pueden ser aplicados a imágenes binarias porque se asume que las imágenes son combinadas por la adición. Esto no es posible en el caso binario, ya que las imágenes binarias solo permiten valores lógicos 0 y 1, por ejemplo 1+1=2, 2 es una valor de pixel que no existe en imágenes binarias. Los operadores morfológicos fueron especialmente diseñados para imágenes binarias, usando un enfoque de teoría de conjuntos.

2.3 La morfología matemática

La principal herramienta usada en toda la tesis es la morfología matemática, los fundamentos matemáticos de la morfología son bastante elaborados así que solo se introducirán los conceptos necesarios para el desarrollo de esta tesis. Se utiliza la morfología matemática como una herramienta para diferentes propósitos, en este proyecto se utilizará para determinar la distribución espacial entre objetos.

2.3.1 Morfología binaria

Morfología Matemática (MM) (Serra, 1982) (Haralick y Shapiro,1992) (Heijmans,1994)(Siole,2003) es el marco para el procesamiento de imágenes basado en teoría de retículas y geometría aleatoria. Esto data desde 1964 cuando fue introducido por Sierra y Matheron (Matheron,1975)(Serra,1982)(Serra,1988). Esta herramienta fue propuesta para analizar estructuras geométricas en imágenes binarias y en niveles de grises. El procesamiento morfológico de imágenes puede simplificar imágenes eliminando objetos irrelevantes, y al mismo tiempo preservando las características de forma esenciales en los objetos.

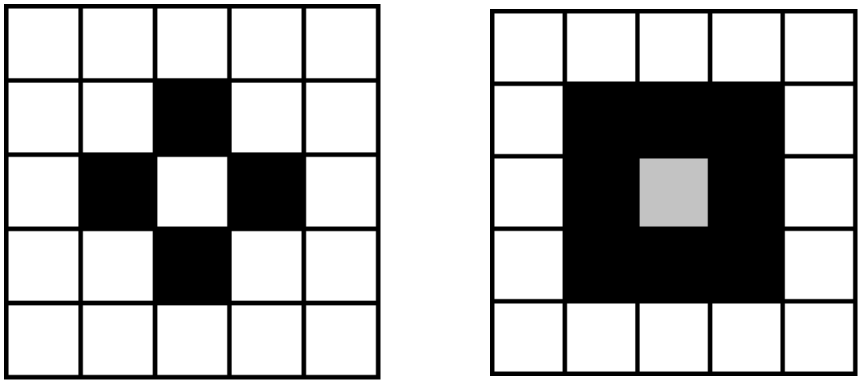
Las imágenes binarias pueden ser descritas en términos de conjuntos de pixeles de un valor constante. Los pixeles de una imagen pueden ser de valor 1 o 0. Esto es, la imagen es únicamente definida para especificar el conjunto:

$$A = \{r | A_f(r) = 1\} \qquad (2.4)$$

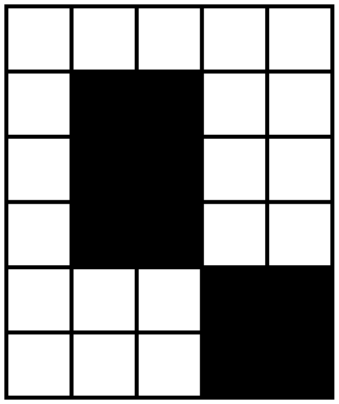
El vector r es la representación de la coordenada (x, y). A es el conjunto, mientras que A_f es la función característica de la imagen binaria que está dada en valores 0 o 1 para especificar el pixel. El término imagen binaria es algunas veces intercambiando por el término conjunto. Los pixeles con valores 1 son los pixeles del primer plano (el o los objetos), mientras que el fondo es de valor 0. Cuando una imagen binaria es desplegada, usualmente el color negro y blanco son el fondo y primer plano respectivamente. Un conjunto que representa una imagen binaria puede consistir en varios objetos. Los objetos son áreas conectadas de elementos con pixeles valor 1, si el fondo tiene elementos con valor 0.

Se trabaja en el espacio discreto, donde una imagen está representada por pixeles que están alineados en una rejilla. Usualmente esta rejilla es rectangular.

Para determinar si dos pixeles son parte del mismo objeto, una regla de conectividad debe ser especificada. Las dos conectividades más comunes son 4 y 8 vecinos, pero también 6 vecinos es posible en una rejilla hexagonal. En el caso de 4 vecinos, un pixel es conectado con otro si el otro pixel es uno de los cuatro vecinos más cercanos a este pixel. Cuando se usa la conectividad de 8 vecinos, entonces también los cuatro vecinos (en las diagonales) de las esquinas más cercanas a las vecindades de estos pixeles son conectadas al pixel. Dos pixeles son partes del mismo objeto si es posible moverse entre los dos pixeles usando un camino de pixeles conectados. En la figura 2-2 estos son uno o dos objetos negros, dependiendo sobre cual conectividad se utilice 8 o 4 vecinos respectivamente. El pixel izquierdo negro y el pixel negro derecho en la figura son solo parte del mismo objeto si es usa la conectividad 8 vecinos.



2-1 a) Conectividad 4 vecinos, b) Conectividad 8 vecinos. El color negro representa los vecinos conectados al pixel gris. (Morales, 2009).



2-2. Dependiendo la conectividad se tienen uno (en 8 vecinos) o dos objetos (en cuatro vecinos). Los objetos están en negro y el fondo en blanco. (Morales, 2009).

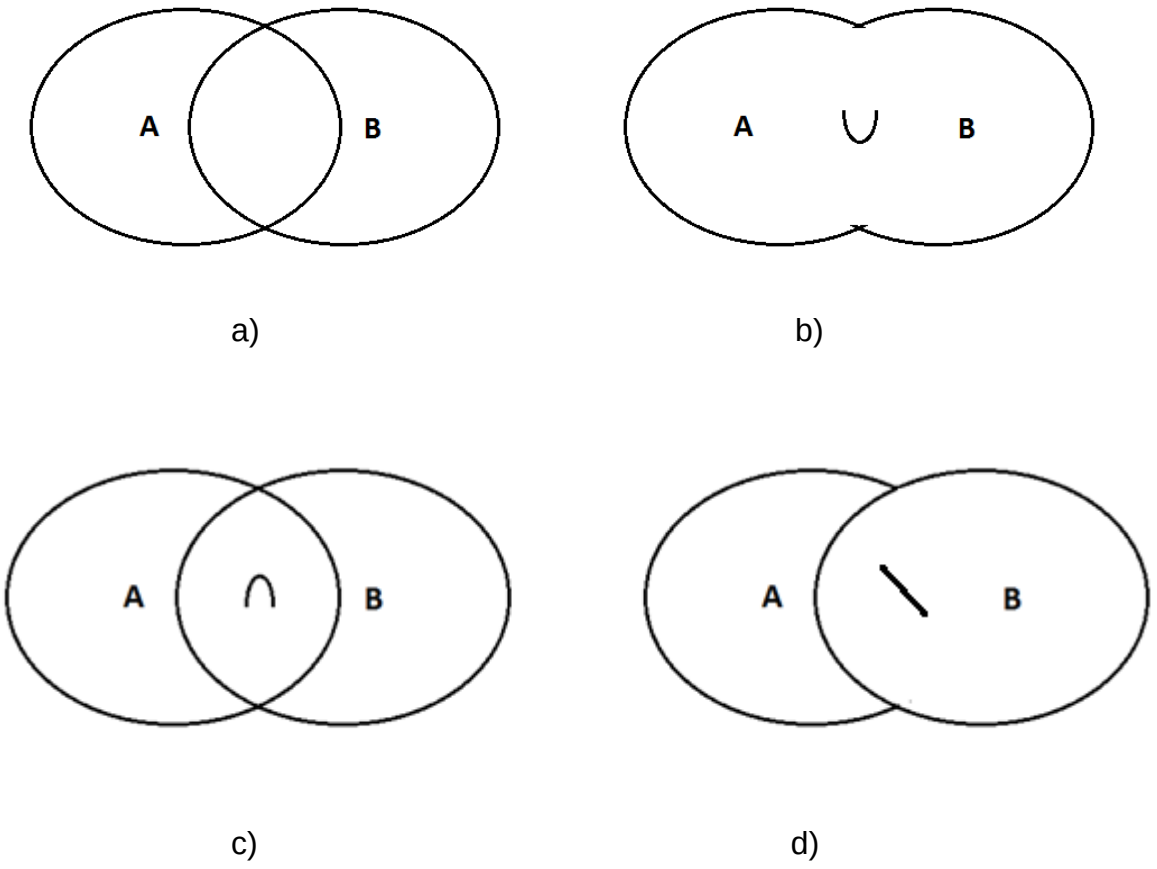
El complemento de un conjunto *A* es definido como:

$$A^C = \{r | A_f(r) = 0\} \qquad (2.5)$$

$$= \{r | r \notin A\}$$

La imagen A^c es obtenida desde A por el cambio de los pixeles negros por pixeles blancos y viceversa.

Se utiliza la operación de conjuntos unión (∪) y la intersección (∩) y diferencia (\), como se ilustra en la Figura 2-3. El conjunto diferencia de A\B es definida por $A \cap B^C$. Un conjunto vacío es representado por símbolo ϕ.



2-3 Operaciones de conjuntos. a) Dos conjuntos A y B, b) Unión de A ∪ B, c) Intersección de los conjuntos A ∩B, d) Diferencia de conjuntos A\B. (Morales, 2009).

La morfología matemática binaria es basada en dos operadores básicos: dilatación y erosión. Estos están definidos en términos del elemento estructural. La ecuación de entrada-salida del procesamiento morfológico de la imagen es denotada por:

$$A' = \psi_B(A) \qquad (2.6)$$

El elemento estructural B es un parámetro importante del operador. Éste es también un conjunto y puede ser pensado como una imagen binaria, aunque es de un tamaño muy pequeño. Por pequeño se entiende una imagen binaria con lados menores a 10 pixeles, mientras la imagen de salida puede tener lados de cientos de pixeles. La función del operador es determinado completamente por B. La forma de este elemento estructural revela que tipo de formas son importantes, las operaciones morfológicas se realizan por lo regular con elementos estructurales cuadrados.

En la siguiente sección se hará uso de los conceptos de traslación y reflexión. La traslación de B por un vector *r* es definido como.

$$T_r(B) = \{b | b - r \in B\} \qquad (2.7)$$

$$= \{b + r | b \in B\}$$

Una imagen digital es siempre descrita en el interior de un área, definida por pixeles horizontales y verticalmente. Esto significa que a los pixeles en el interior del área se les asigna un valor, pero los valores de los pixeles exteriores son desconocidos. Una traslación de las coordenadas de un determinado pixel puede referirse a un pixel con un valor indefinido. Esto debe ser tomado en cuenta.

Todos los objetos entonces deben estar dentro del área viable de la imagen. Los pixeles fuera del área de la imagen se le puede asignar el valor 0, que es también llamado cero de relleno. Esto tiene la desventaja, de cortar abruptamente los objetos que tocan el borde de la imagen. Se puede generalizar este relleno para otros valores, por ejemplo, para asignar el valor máximo de los pixeles anteriores.

Otra posibilidad es el borde espejo. Los bordes de la imagen funcionan como líneas espejo, y los valores de pixel del área visible de la imagen son reflejados. Cambios abruptos de los bordes de la imagen son evitados de esta manera, pero nuevos valores no necesariamente representan el valor real de los pixeles indefinidos.

Ambos métodos son visualizados en la figura 2-4 donde P_{XY} es el valor del pixel. Para la dilatación y la erosión, es necesario asignar un valor a los bordes como se verá más adelante (Morales, 2009).

0	0	0	P_{11}	P_{12}	P_{13}
0	0	0	P_{21}	P_{22}	P_{23}
0	0	0	P_{31}	P_{32}	P_{33}
0	0	0	P_{41}	P_{42}	P_{43}
0	0	0	P_{51}	P_{52}	P_{53}
0	0	0	P_{61}	P_{62}	P_{63}

a)

P_{13}	P_{12}	P_{11}	P_{11}	P_{12}	P_{13}
P_{23}	P_{22}	P_{21}	P_{21}	P_{22}	P_{23}
P_{33}	P_{32}	P_{31}	P_{31}	P_{32}	P_{33}
P_{43}	P_{42}	P_{41}	P_{41}	P_{42}	P_{43}
P_{53}	P_{52}	P_{51}	P_{51}	P_{52}	P_{53}
P_{63}	P_{62}	P_{61}	P_{61}	P_{62}	P_{63}

b)

2-4. Pixeles en los bordes de la imagen. a) Relleno con cero, b) Espejo de bordes. Pixeles fuera del área visible se les asigna valores por omisión (Morales, 2009)

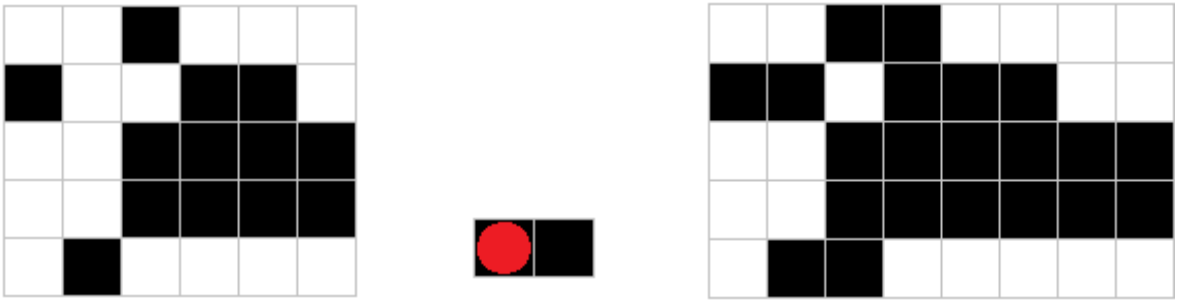
2.3.1.1 Dilatación binaria

La dilatación binaria, combina dos conjuntos para usar el vector de adición de un conjunto de elementos. Esta adición es llamada la edición de Minkowski. La dilatación se define como:

$$A \oplus B = \{a + b : a \in A, b \in B\} \qquad (2.8)$$

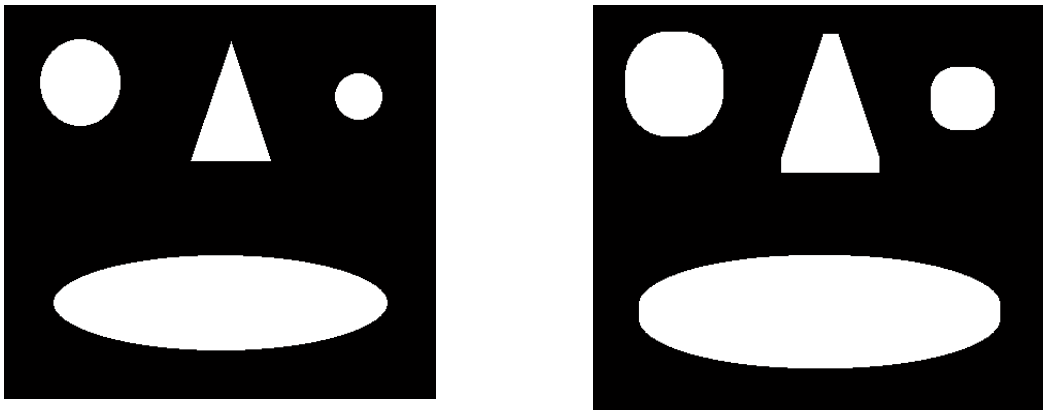
La adición de Minkowski es conmutativa, la dilatación es también conmutativa, esto es $A \oplus B = B \oplus A$. En la práctica A es una imagen y B es el elemento estructural pequeño y la operación es siempre descrita como la dilatación de la imagen por el elemento estructural.

La figura 2.5 muestra el efecto de la dilatación morfológica. La dilatación agrega pixeles al conjunto A. El origen del elemento estructural es posicionado para cada pixel del objeto. Cada pixel que es ahora parte del elemento estructural, será parte de la imagen dilatada.



a) A b)B c)$A \oplus B$

2-5. Dilatación binaria. Los elementos oscuros representan el objeto. a) Imagen original, b) Elemento estructural con origen ⊗,c) Dilatación de A por el elemento estructural B.



a) Imagen Original b)$A \oplus B$

2-6. Dilatación binaria considerando B un elemento estructural cuadrado, los blancos son los pixeles del objeto y el negro representa el fondo, a) Imagen Original, b) Imagen Dilatada.

Una dilatación agranda los objetos en la imagen, para incrementar el número de pixeles en la imagen. En la Figura 2-6 se muestra el dilatado de una imagen binaria (los pixeles blancos son parte del conjunto) por un elemento estructural cuadrado con tamaño 4. Note que además de ampliar el objeto, los agujeros son rellenados y las líneas del contorno aparecen más nítidas.

La dilatación tiene algunas propiedades interesantes. La primera de todas, es que es distributiva con respecto a la operación de unión. Es también invariante a la traslación, como esta es una propiedad del procesamiento de imágenes, de acuerdo a la ecuación (2.3). Las propiedades distributiva e invariante a la traslación para la dilatación son respectivamente:

$$(A_1 \cup A_2) \oplus B = (A_1 \oplus B) \cup (A_2 \oplus B) \qquad (2.9)$$

$$A \oplus T_r(B) = T_r(A \oplus B) \qquad (2.10)$$

El operador es también extensivo, lo que significa que el conjunto inicial es parte del dilatado del conjunto:

$$\forall A: A \subseteq A \oplus B \qquad (2.11)$$

Este es el caso cuando el origen es parte del elemento estructural $(0 \in B)$.

En los ejemplos mostrados, el efecto de los bordes debe tomarse en cuenta. Para la dilatación, todos los pixeles fuera de la imagen son puestos en 0. De esta manera solo los pixeles del objeto son visibles. Por otro lado, los pixeles del borde de la imagen pueden introducir imprevistos no deseados en la imagen.

2.3.1.2 Erosión Binaria

La erosión binaria es morfológicamente el dual con respecto a la complementación de la dilatación. La erosión está definida en términos de la sustracción de Minkowski:

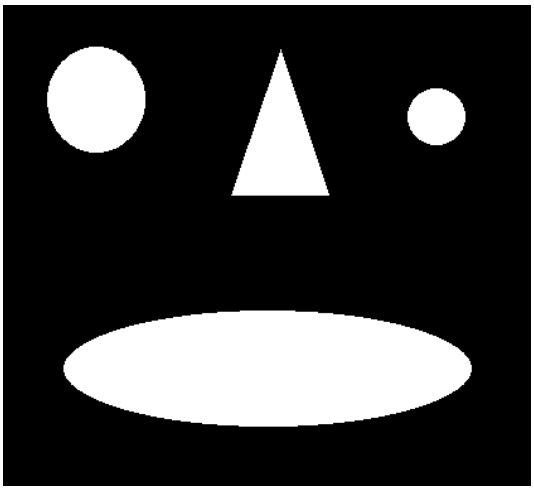
$$A \ominus B = \{r | r + b \in A, \forall b \in B\} \qquad (2.12)$$

A diferencia de la dilatación la erosión no es conmutativa. Como con la dilatación, entonces es posible reescribir la erosión, esta vez usando la operación intersección.

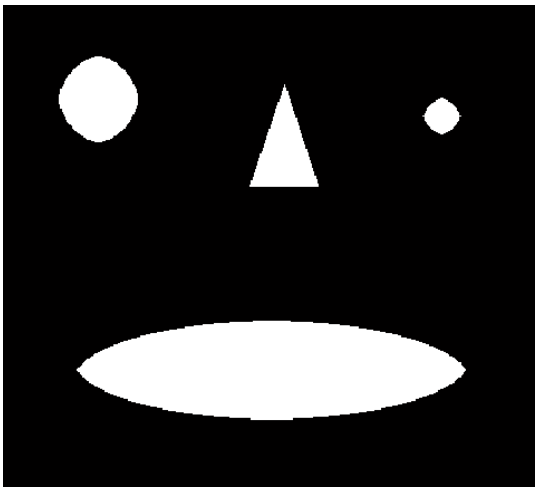
$$X \ominus B = \bigcap_{b \in B} X_b = \{x \in E: B_x \subseteq X\} \qquad (2.13)$$

a) A b)B c)$A \ominus B$

2-7. Erosión binaria. Los elementos oscuros representan el objeto. a) Imagen original, b) Elemento estructural con origen ⊗, (c) erosión de A por el elemento estructural B.



a) Imagen Original



b)$A \ominus B$

2-8. a) Imagen original, b) Erosión binaria por B, con B un elemento estructural cuadrado, los blancos son los pixeles del objeto y el negro representa el fondo.

Un ejemplo de la erosión se muestra en la Figura 2-7. El elemento estructural es trasladado para cada pixel de la imagen, uno a la vez. Si cada pixel que es parte del elemento estructural trasladado es también parte del objeto, entonces el pixel en la ubicación del elemento estructural sea parte de la imagen erosionada.

Una erosión reduce los objetos; también los objetos pequeños desaparecen y los objetos conectados por pequeños puentes tienden a desaparecer (Figura 2-8).

La erosión también cumple las propiedades distributiva e invariante a la traslación, lo cual hace el procesamiento de imagen de acuerdo a la definición de la sección anterior. Las propiedades distributiva e invariante a la traslación se muestran a continuación:

$$(A_1 \cap A_2) \ominus B = (A_1 \ominus B) \cap (A_2 \ominus B) \qquad (2.14)$$

$$A \ominus T_r(B) = T_{-r}(A \ominus) B \qquad (2.15)$$

La erosión es creciente:

$$A_1 \subseteq A_2 \Rightarrow A_1 \ominus B \subseteq A_2 \ominus B \qquad (2.16)$$

La erosión es llamado el operador anti-extensivo si para todas las imágenes, la erosión de la imagen de entrada es parte de la misma imagen de salida.

$$\forall A: A \ominus B \subseteq A \qquad (2.17)$$

Este es el caso cuando el origen es parte del elemento estructural $(0 \in B)$.

En los ejemplos dados, el efecto del borde debe ser tomado en cuenta. Para la erosión, todos los pixeles anteriores de la imagen son el máximo valor que puede tomar el conjunto, tal que para imagen binarias el valor es 1. De esta manera, solo los pixeles predefinidos contribuyen a la erosión. Los pixeles del borde son entonces parte de un objeto grande fuera del área visible. Esto podría dar lugar para preservar estos pixeles del borde, mientras que esto no necesariamente es deseado. Los bordes espejo podrían ser usados pero esto generalmente no se hace. Rellenar los borde asegura que ningún pixel fuera del a imagen contribuye a la dilatación o a la erosión.

2.3.1.4 Apertura y cerradura binaria

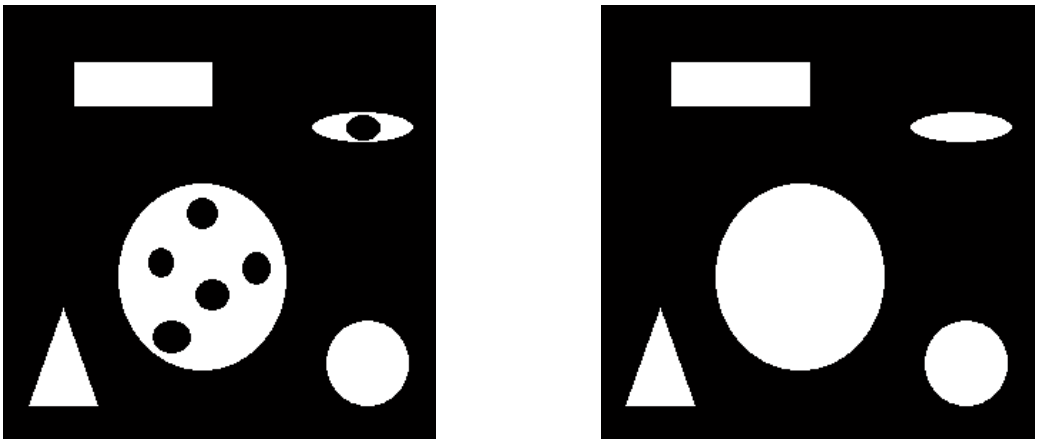
La dilatación y erosión son bloques de construcción primario para otros operadores morfológicos. Los operadores básicos pueden ser combinados de diferentes maneras. La combinación simple es el encadenamiento de un operador básico con otro. Esta combinación forma operadores morfológicos secundarios.

Cerradura

La cerradura morfológica está definida como una dilatación seguida por una erosión:

$$A \cdot B = (A \oplus B) \ominus B \qquad (2.18)$$

Como se ve el mismo elemento estructural *B* es usado para la dilatación y la erosión. Las operaciones morfológicas se han definido con la notación de Minkowski. La erosión compensa parcialmente el efecto de la dilatación pero este efecto es parcialmente desecho por la erosión. También el crecimiento de los objetos es cancelado por la disminución que se produce en el paso de la erosión. Generalmente une separaciones estrechas y entrantes delgadas, así como elimina pequeños huecos y rellena agujeros del contorno. La cerradura es entonces un filtro suavizador. La figura 2-9 muestra un ejemplo.



a) A b) $A \cdot B$

2-9. a) Imagen original, b) Cerradura binaria por un elemento estructural B tipo cuadrado. Donde los pixeles color blanco representan los objetos y los negros el fondo.

Cada cerradura es creciente, extensiva e idempotente, como se expresa a continuación:

$$A_1 \subseteq A_2 \Rightarrow A_1 \cdot B \subseteq A_2 \cdot B \qquad (2.19)$$

$$A \subseteq A \cdot B \qquad (2.20)$$

$$A \cdot B = (A \cdot B) \cdot B \qquad (2.21)$$

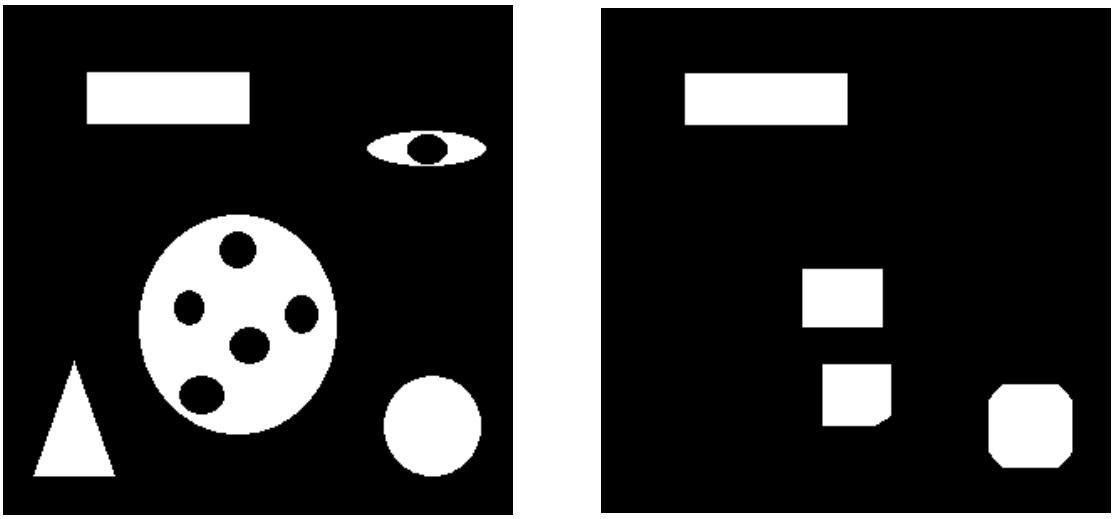
La idempotencia es una propiedad importante. La idempotencia quiere decir que si se aplica una segunda cerradura con el mismo elemento estructural producirá la misma imagen, como si solo se aplicara la cerradura una sola vez.

Apertura

La apertura morfológica está definida como una erosión seguida por una dilatación:

$$A \circ B = (A \ominus B) \oplus B \qquad (2.22)$$

La dilatación tratará de eliminar la operación de erosión pero los objetos que desaparecen a causa de la erosión no pueden ser recobrados por la dilatación. Al igual que la cerradura, la apertura es un filtro que suaviza la imagen. Un ejemplo de la apertura binaria se muestra en la Figura 2-10.



a) A $A \circ B$

2-10. a) Imagen Original, b) Apertura binaria por un elementó estructural *B* tipo cuadrado. Donde los pixeles blanco representan los objetos y los negros el fondo.

Todas las aperturas son crecientes, anti-extensivas, idempotentes como se expresa a continuación:

$$A_1 \subseteq A_2 \Rightarrow A_1 \circ B \subseteq A_2 \circ B \quad (2.23)$$

$$A \circ B \subseteq A \quad (2.24)$$

$$A \circ B = (A \circ B) \circ B \quad (2.25)$$

2.3.2 Morfología en escala de grises

La teoría de morfología matemática fue desarrollada para ser aplicada a imágenes binarias. Los operadores morfológicos binarios se pueden considerar operadores lógicos que trabajan sobre imágenes lógicas: un punto en el espacio, tal que un pixel en una imagen discreta es parte del conjunto o no lo es, es decir, que un pixel tiene un valor lógico e 1 o 0.

En las imágenes en niveles de grises es diferente, mientras una imagen binaria es representada por los colores blanco y negro, una imagen en nivel de gris consiste de diferentes tonos de gris. En procesamiento de imágenes el rango de valores para imágenes en escala de grises es entre los valores de 0 y 255, tomando solo valores enteros, siendo 0 el tono más oscuro(negro) y 255 el tono más claro(blanco). Esto es para imágenes en 8 bits, lo que brinda 256 posibles valores.

2.3.2.1 Dilatación en niveles de grises

Para imágenes en niveles de grises, si *B* es el elemento estructural, la dilatación de la función *f* dentro de la ventana de observación definida por *B* desplazado de manera que el origen de *B* está centrado en *x*

$$\delta_f(x) = max(x_k, k \epsilon B) \quad (2.26)$$

Donde x_k es el valor que toma la imagen en el punto *x*+*k*, es decir f(*x*+*k*).

2.3.2.2 Erosión en nivel de gris

Para la erosión se tiene la siguiente fórmula:

$$\varepsilon_f(x) = min(x_k, k \epsilon B) \quad (2.27)$$

Las propiedades más importantes de la dilatación y la erosión son que ambas transformaciones son crecientes y que si el origen del elemento estructural está contenido en él mismo, entonces, son extensivas y anti-extensivas respectivamente. Ambas funciones son además duales, de manera que:

$$\delta_B(f)^C = \varepsilon_B(f^C) \quad (2.28)$$

Tanto la erosión como la dilatación no cumplen con la propiedad de idempotencia. Al aplicar sucesivamente la erosión el nivel de gris de la imagen disminuye.

$$\delta_B(f) \neq \delta_B\big(\delta_B(f)\big), \varepsilon_B(f) \neq \varepsilon_B\big(\varepsilon_B(f)\big) \quad (2.30)$$

2.3.2.3 Apertura y Cerradura en niveles de gris

Como ya se vio anteriormente al aplicar una erosión seguida de una dilatación o viceversa, se obtiene una apertura y una cerradura respectivamente.

Apertura. La apertura de una función por un elemento estructural *B*, se denota por:

$$\gamma_B(f) = \delta_B\big(\varepsilon_{\check{B}}(f)\big) \quad (2.31)$$

Si el elemento estructural es simétrico, entonces la transposición del elemento estructural no produce ningún cambio y se puede re-escribir de la siguiente forma:

$$\gamma_B(f) = \delta_B\big(\varepsilon_B(f)\big) \quad (2.32)$$

La apertura cumple, entre otras, las siguientes propiedades:

Creciente:

$$f \leq g \Leftrightarrow \gamma(f) \leq \gamma(g) \qquad (2.33)$$

Anti-extensiva:

$$\gamma(f) \leq f \qquad (2.34)$$

Cerradura. La cerradura de una función *f* por un elemento estructural *B*, denotado por $\varphi_B(f)$, se define como:

$$\varphi_B(f) = \varepsilon_B\left(\delta_{\check{B}}(f)\right) \qquad (2.35)$$

Si el elemento estructural es simétrico, entonces la transposición del mismo no produce ningún cambio y se puede poner que $\varphi_B(f) = \varepsilon_B\left(\delta_B(f)\right)$.

La cerradura cumple, entre otras, las siguientes dos propiedades:

Creciente:

$$f \leq g \Leftrightarrow \varphi(f) \leq \varphi(g) \qquad (2.36)$$

Extensiva:

$$\varphi_B(f) \geq f \qquad (2.37)$$

Como consecuencia, se puede establecer el siguiente ordenamiento parcial:

$$\gamma(f) \leq f \leq \varphi(f) \qquad (2.38)$$

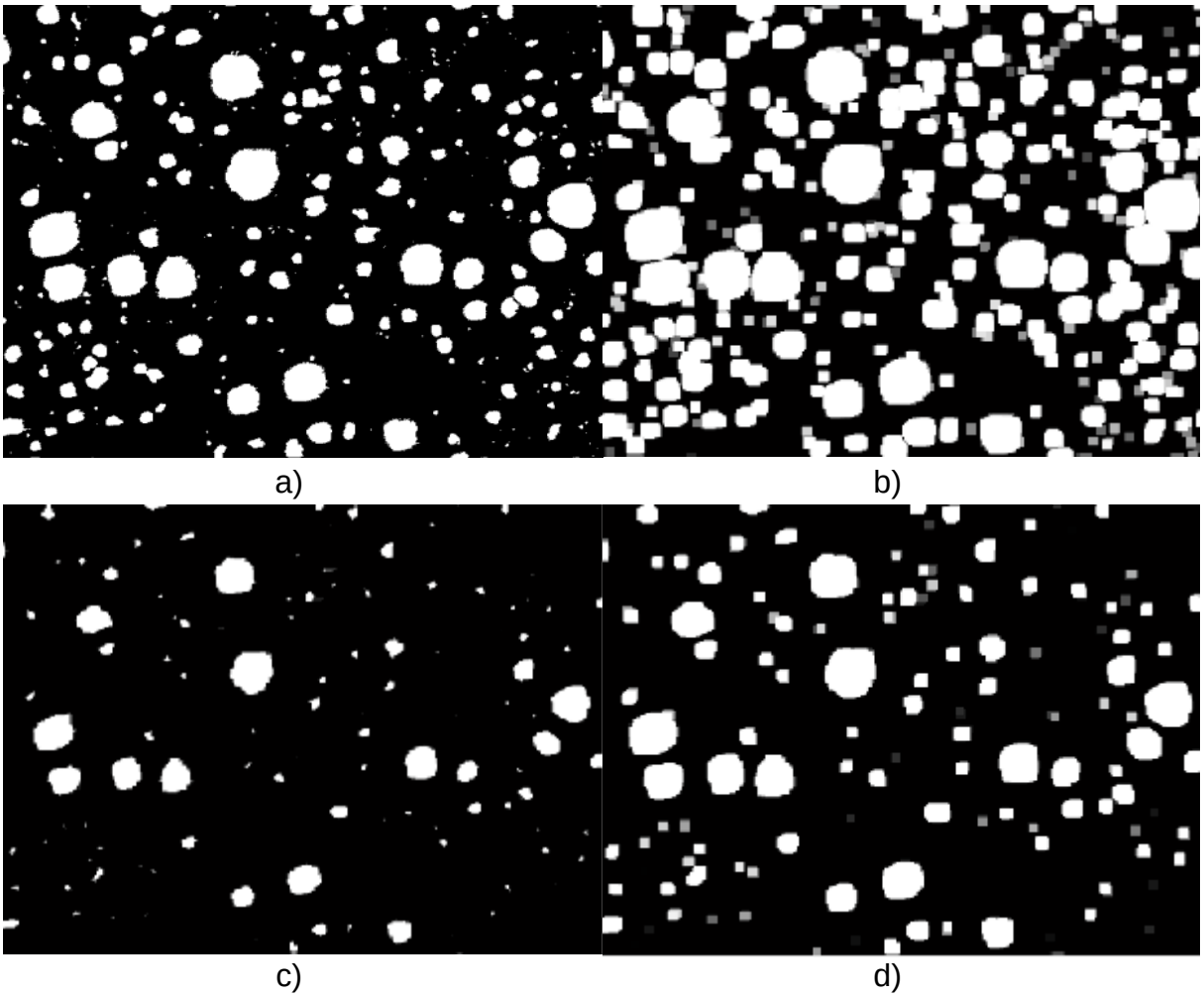
De la misma manera que las operaciones erosión-dilatación son operaciones duales también lo son la apertura y cierre, de la forma siguiente:

$$\gamma_B(f)^C = \varphi_B(f^C) \qquad (2.39)$$

A diferencia de la erosión y la dilatación, la apertura y la cerradura son idempotentes, es decir:

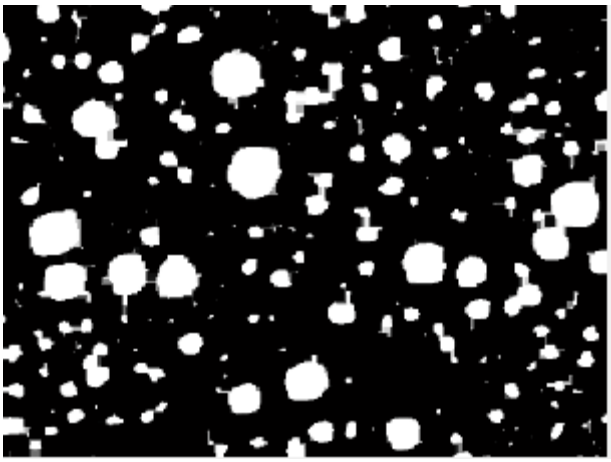
$$\gamma\gamma = \gamma \text{ y } \varphi\varphi = \varphi \qquad (2.40)$$

En la Figura 2-11, se muestra los cuatro operadores en una imagen en niveles de grises. El crecimiento o reducción de los objetos de la imagen y el llenado de hoyo o la desconexión de líneas delgadas es hora trasladado al dominio de niveles de grises. Una dilatación hace más brillosa una imagen mientras que la erosión oscurece la imagen. Un hueco en una imagen en niveles de grises es un área oscura pequeña alrededor de valores de gris claro. Una cerradura puede entonces incrementar el valor del nivel de gris de los pixeles del hueco, pasando lo opuesto con la apertura (Morales, 2009).



a) b)

c) d)



e)

2-11. a) Imagen Original, b) Dilatacion 3, c) Erosion 3 ,d) Apertura 3, e) Cerradura 3.

2.4 Otro método para obtener los operadores morfológicos de erosión y dilatación.

Existe otro método que permite obtener los operadores morfológicos de erosión y dilatación de una imagen procesada en software de una manera más eficiente si se habla de tiempo. Dicho método consiste en hacer cuatro diferentes procesamientos básicos a la imagen a procesar, y es equivalente a utilizar una máscara o kernel de tamaño 3x3, con todos sus elementos con valor 1; dichos procesamientos son llamados barrido izquierda-derecha, arriba-abajo, derecha-izquierda y abajo-arriba, en los siguientes apartados se explica en que consiste cada uno de ellos.

2.4.1 Procesamiento izquierda-derecha.

Este procesamiento a la imagen consiste en lo siguiente y como su nombre lo indica, se empieza a procesar la imagen de izquierda a derecha, como primer paso se toman los valores de dos pixeles, estos son comparados y dependiendo que operador morfológico se quiera calcular dilatación o erosión, se decide cual dato posee el valor de mayor o menor intensidad respectivamente; una vez que se tiene el dato de interés este valor sustituye al valor del primer pixel que fue leído.

Para comprender mejor el procesamiento se explica a continuación con un ejemplo.

La imagen que se analiza en este ejemplo es de seis renglones por cinco columnas (6X5), en la figura 2-12.1 se muestra la imagen con sus respectivos valores de direcciones e intensidades de cada pixel.

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

a) Direcciones de la imagen

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

b) Valor de intensidad en cada dirección (pixeles)

2-12.1 a) Direcciones en imagen de 6x5, b) Intensidad de los pixeles.

El procesamiento que se debe realizar es el siguiente, tomar el valor de la dirección 0 y dirección 1, comparar los valores obtenidos de dichas direcciones y decidir cuál posee la mayor o menor intensidad (dilatación, erosión), una vez que se tiene el resultado de dicha comparación es necesario escribir el valor obtenido de la comparación en la dirección del primer dato obtenido. Es decir:

Direcciones	Valores	Dato Mayor	Dirección de almacenamiento de resultado
0,1	110,100	110	0
1,2	100,20	100	1
2,3	20,3	20	2
3,4	3,40	40	3
5,6	45,60	60	5
6,7	60,7	60	6
7,8	7,80	80	7
8,9	80,90	90	8
.	.	.	.
.	.	.	.
27,28	27,280	280	27
28,29	289,290	290	28

2.4.1 - Procesamiento a realizar a la imagen (izquierda-derecha).

Como se observa al realizar el procesamiento izquierda-derecha la última columna de datos no es afectada.

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

a)

110	100	20	40	40
60	60	80	90	90
110	120	120	14	14
160	160	180	190	190
210	220	220	24	24
250	27	280	290	290

b)

2-12.2 a) Imagen Original, b) Imagen procesada izquierda-derecha.

2.4.2 Procesamiento arriba-abajo

Observando nuevamente la Figura 2-12.1 a) y b).

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

a) Direcciones de la imagen

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

b) Valor de intensidad en cada dirección (pixeles)

El procesamiento en este caso es el siguiente:

Direcciones	Valores	Dato Mayor	Dirección de almacenamiento de resultado
0,5	110,45	110	0
5,10	45,100	100	5
10,15	100,150	150	10
15,20	150,200	200	15
20,25	200,250	250	20
1,6	100,60	100	1
6,11	60,110	110	6
11,16	110,160	160	11
16,21	160,210	210	16
21,26	210,26	210	21
.	.	.	.
.	.	.	.
19,24	190,24	190	19
24,29	24,290	240	24

2.4.2. Procesamiento a realizar en la imagen (Arriba-abajo)

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

a)

110	100	20	80	90
100	110	120	80	90
150	160	120	180	190
200	210	220	280	290
250	210	220	280	290
250	26	27	280	290

b)

2-12.3 a) Imagen Original, b) Imagen procesada arriba-abajo.

Como se observa al realizar el procesamiento arriba-abajo el último renglón no es afectado.

2.4.3 Procesamiento derecha-izquierda

Observando nuevamente la Figura 2-12.1 a) y b).

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

a) Direcciones de la imagen

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

b) Valor de intensidad en cada dirección (pixeles)

El procesamiento a realizar en este caso es el siguiente:

Direcciones	Valores	Dato Mayor	Dirección de almacenamiento de resultado
4,3	40,3	40	4
3,2	3,20	20	3
2,1	20,100	100	2
1,0	100,110	110	1
9,8	90,80	90	9
8,7	80,7	80	8
7,6	7,60	60	7
6,5	60,45	60	6
.	.	.	.
.	.	.	.
29,28	290,280	290	29
28,27	280,27	280	28
27.26	27,26	27	27
26,25	26,250	250	26

2.4.3. Procesamiento a realizar en la imagen (derecha-izquierda)

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

a)

110	110	100	20	40
45	60	60	80	90
100	110	120	120	14
150	160	160	180	190
200	210	220	220	24
250	250	27	280	290

b)

2-12.4 a) Imagen Original, b) Imagen procesada derecha-izquierda.

Como se observa al realizar el procesamiento derecha-izquierda la primera columna de la imagen no es afectada.

2.4.4 Procesamiento abajo-arriba.

Observando nuevamente la Figura 2-12.1 a) y b).

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

c) Direcciones de la imagen

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

d) Valor de intensidad en cada dirección (pixeles)

El procesamiento a realizar en este caso es el siguiente:

Direcciones	Valores	Dato Mayor	Dirección de almacenamiento de resultado
25,20	250,200	250	25
20,15	200,150	200	20
15,10	150,100	150	15
10,5	100,45	100	10
5,0	45,110	110	5
26,21	26,210	210	26
21,16	210,160	210	21
16,11	160,110	160	16
11,6	110,60	110	11
6,1	60,100	100	6
.	.	.	.
.	.	.	.
19,14	190,14	190	19
14,9	14,90	90	14
9,4	90,40	90	9

2.4.4. Procesamiento a realizar en la imagen (abajo-arriba)

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

c)

110	100	20	3	40
110	100	20	80	90
100	110	120	80	90
150	160	120	180	190
200	210	220	180	190
250	210	220	280	290

d)

2-12.4 a) Imagen Original, b) Imagen procesada abajo-arriba.

Como se observa al realizar el procesamiento abajo-arriba el primer renglón de la imagen no es afectado.

2.4.5 Operadores morfológicos de erosión y dilatación utilizando el segundo método planteado.

Una vez explicados los cuatro procesamientos anteriores, lo que prosigue para obtener el operador morfológico de erosión y dilatación, es llevar a cabo los procesos en el siguiente orden: izquierda-derecha, arriba-abajo, derecha-izquierda y abajo-arriba.

Si se desea obtener el operador morfológico erosión se busca el dato menor en los cuatro procesamientos y en caso de que se desee el operador morfológico dilatación se busca el dato mayor en éstos.

En la Figura 2-12.5 se muestran los resultados obtenidos al aplicar dichos procesos en el orden citado anteriormente.

110	100	20	3	40
45	60	7	80	90
100	110	120	13	14
150	160	17	180	190
200	210	220	23	24
250	26	27	280	290

a)

45	7	3	3	3
45	7	3	3	3
45	7	7	7	13
100	17	13	13	13
26	17	17	17	23
26	26	23	23	23

b)

110	110	100	90	90
110	120	120	120	90
160	160	180	190	190
210	220	220	220	190
250	250	280	290	290
250	250	280	290	290

c)

2-12.5 a) Imagen original, b) Imagen erosionada, c) Imagen dilatada

2.5 Granulometría

Una granulometría es el proceso de tamizado a través de diferentes mallas de tamiz que se incrementa en tamaño. Los objetos más pequeños se filtran primero, posteriormente elementos cada vez más grandes se van filtrando, hasta el final se remueven los elementos más grandes, los tamices son capaces de clasificar los objetos de la imagen en función de su tamaño (Soille, 2003).

El residuo después del tamizado es parte de la muestra de entrada.

Si se tamiza repetidamente usando el mismo tamaño de malla entonces no filtrarán nuevos objetos.

Estas propiedades son de la apertura $\gamma_n(A)$(para imágenes binarias y en niveles de gris):

Creciente: $A_1 \subseteq A_2 \Rightarrow \gamma_1(A) \subseteq \gamma_2(A)$

Anti-extensiva: $\gamma_B(A) \subseteq A$

Idempotente: $\gamma_B(A) = \gamma_B\big(\gamma_B(A)\big)$

Si se utilizan dos tamices de diferentes tamaños de malla, el residuo que se obtiene es de un tamaño más grande que el tamaño de la malla de tamiz que se utilizó. A esto se le llama la propiedad de absorción y en el caso de la apertura esta se formaliza como lo indica la ecuación 2.41.

$$\gamma_B(\gamma_C(A)) = \gamma_C(\gamma_B(A)) = \gamma_{\max(B,C)}(A) \tag{2.41}$$

Donde el max(B,C) significa el elemento estructural que es superconjunto de otro elemento estructural. Lo anterior es equivalente a:

$$B \subseteq C \Rightarrow \gamma_B(A) \supseteq \gamma_C(A) \tag{2.42}$$

La curva granulométrica se define como:

$$CG_{nB}(A) = vol(\gamma_{nB}(A)), n \geq 0. \tag{2.43}$$

El índice *n* incrementa el tamaño del elemento estructural. Donde *vol* es la suma de todos los niveles de gris de la imagen. La curva granulométrica normalizada es una curva granulométrica pero con los resultados escalados (divididos) por la suma de los niveles de gris de la imagen original.

$$CGN_{nB}(A) = \frac{vol(\gamma_{nB}(A))}{vol(A)}, n \geq 0. \tag{2.44}$$

En la práctica lo que se ven en la curva es la salida de tamiz, no el residuo de este. A esto se le llama espectro de tamaño o espectro de patrones (ET o EP) o distribución de tamaños.

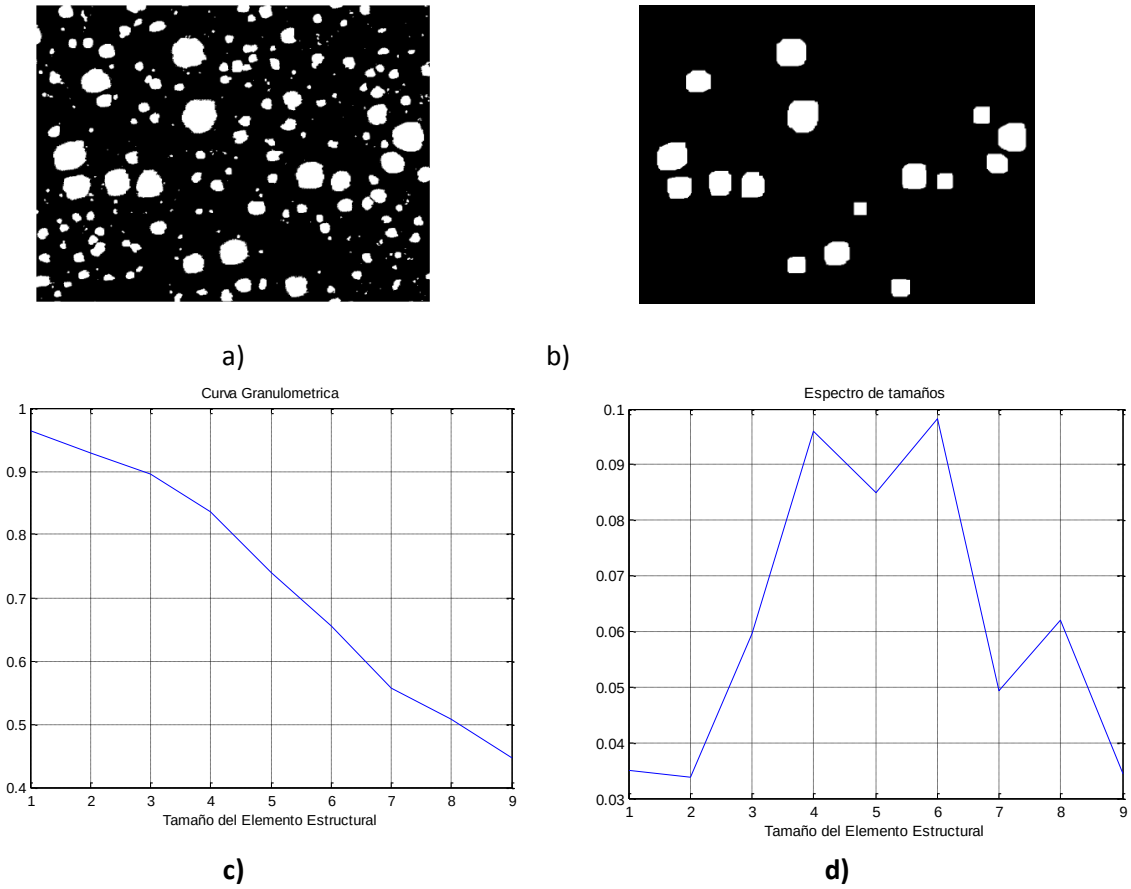
$$ET_{nB}(A) = vol\left(\gamma_{nB}(A) - \gamma_{(n+1)B}(A)\right), n \geq 0 \tag{2.45}$$

Donde la substracción "-" es la diferencia de los valores de nivel de gris de los pixeles. En el caso binario, la interpretación del espectro de tamaños es muy sencilla, el valor del elemento estructural *n* en el histograma indica entonces la cantidad de pixeles que han sido obtenidos entre la apertura por un elemento estructural *nB* y el elemento estructural *(n+1)B*.

Se define el espectro de tamaños normalizados (ETN), igualmente como para la curva granulométrica (tal que se divide el resultado por la suma de los niveles de gris de la imagen original). Pudiéndose definir una normalización del elemento estructural ET:

$$ETN_{(nB)}(A) = \frac{vol\left(\gamma_{nB}(A) - \gamma_{(n+1)B}(A)\right)}{vol(nB)}, n \geq 0 \quad (2.46)$$

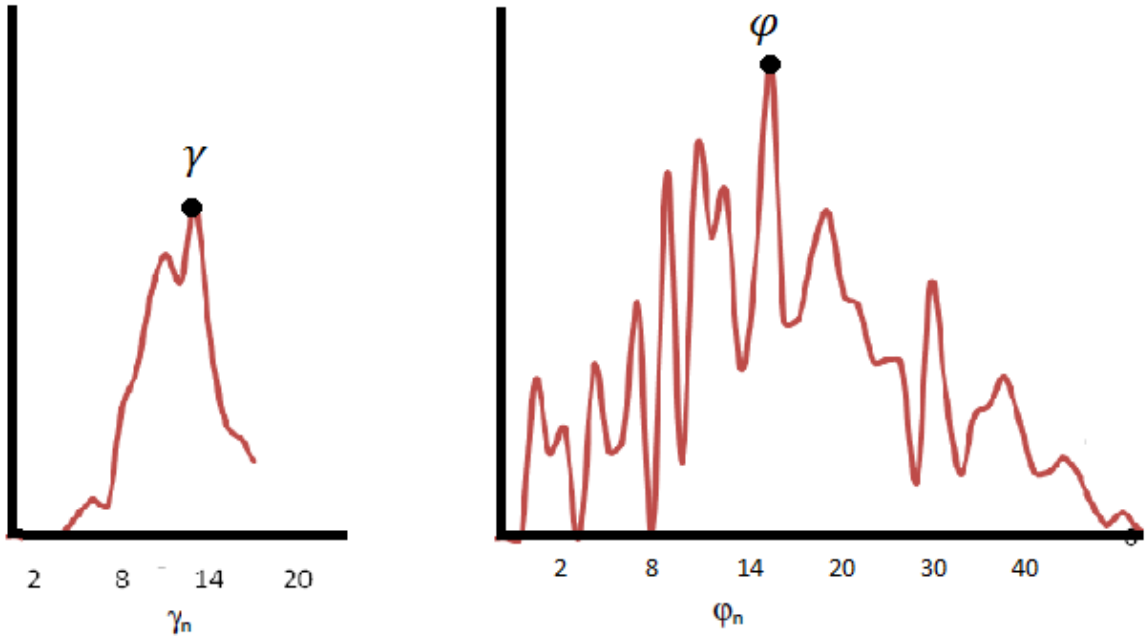
Entonces la distribución de tamaños índica el número de los objetos que son tamizados asumiendo que los objetos tienen el mismo tamaño y forma como la del elemento estructural *nB*.



2-13 a) Imagen Original, b) Imagen con apertura 10, c) Curva granulométrica, d) Espectro de tamaño obtenido

2.6 Índice de cuantificación espacial.

Gracias a la apertura en niveles de gris, es posible calcular los tamaños de los nódulos en la imagen a analizar, ahora bien si se aplica el operador morfológico de cerradura en niveles de grises es posible encontrar la distancia máxima y mínima que existe entre los nódulos, y así, una vez obteniendo los valores máximos tanto para el tamaño del nódulo(aperturas) y distancias entre nódulos (cerraduras) hacer el cálculo de un índice que indique la distribución espacial entre objetos, en este caso entre los nódulos.



2-14 Espectro de tamaños obtenidos con aperturas y cerraduras.

Dicho índice será calculado de la siguiente manera:

$$I = \frac{\gamma max}{\varphi max} \tag{2.47}$$

Dónde:

γmax es el valor máximo calculado en el espectro de tamaño usando aperturas.

φmax es el valor máximo calculado en el espectro de tamaño usando cerraduras.

2.7 FPGA

Los FPGA (Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo), introducidas por Xilinx en 1985, son los dispositivo programables más generales para el usuario. También se denominan LCA (Logic Cell Array, Arreglo de compuertas programables en campo). Consisten en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad lo que se programa en un FPGA son los

conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques.

A continuación se citan algunas de las ventajas del porque trabajar con las tarjetas FPGA:

- Rompen el paradigma de ejecución secuencial logrando más en cada ciclo de reloj y a nivel hardware ofrece tiempos de respuesta más veloces.
- La tecnología FPGA ofrece flexibilidad, capacidades de rápido desarrollo de prototipos para enfrentar los retos de que un producto se libere tarde al mercado. Se puede probar una idea o un concepto y verificando en hardware sin tener que pasar por el proceso de fabricación por el que pasa un diseño personalizado ASIC (Application Specific Integrated Circuit).
- Se puede realizar cambios en un diseño al instante.

Con ayuda de una herramienta de diseño (software *Active HDL o Quartus*) se realizará la descripción de los algoritmos a implementar en el FPGA, y con éste mismo se realizarán las simulaciones de las descripciones para comprobar su funcionamiento.

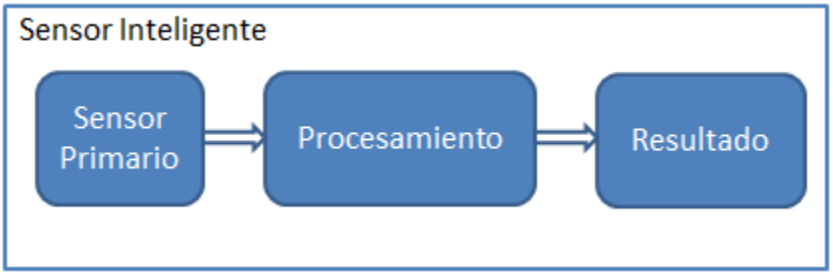
Capítulo 3.
Metodología

En este capítulo se presenta los pasos a seguir para el desarrollo de este trabajo, el software utilizado para el desarrollo del mismo y los módulos desarrollados para la implementación en el FPGA.

3.1 Sensor Inteligente

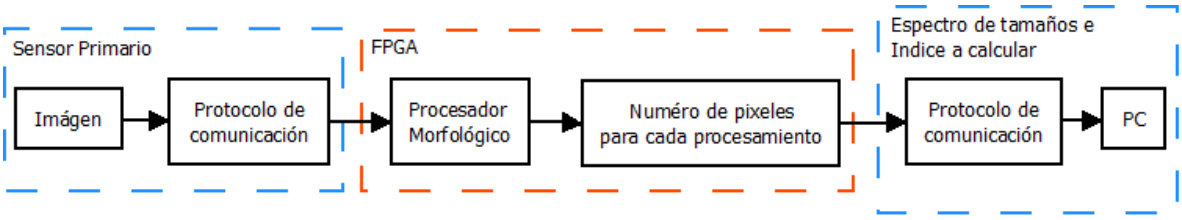
Uno de objetivos principales en este trabajo es el desarrollo de un sensor inteligente el cual tenga la capacidad de capturar una imagen, procesarla y arrojar un resultado, en este caso, un índice con el cual se pueda analizar algunas de las propiedades de los metales.

Un sensor inteligente es aquel dispositivo conformado por un sensor primario, una etapa de procesamiento y otra etapa donde se arroja el resultado obtenido (Figura 3-1), el sensor primario es el encargado de hacer la adquisición de los datos que se desean analizar, la etapa de procesamiento es donde se aplican los algoritmos o procesos necesarios a los datos capturados para así tener al final un resultado de dicho análisis.



3-1 Diagrama general de un sensor inteligente

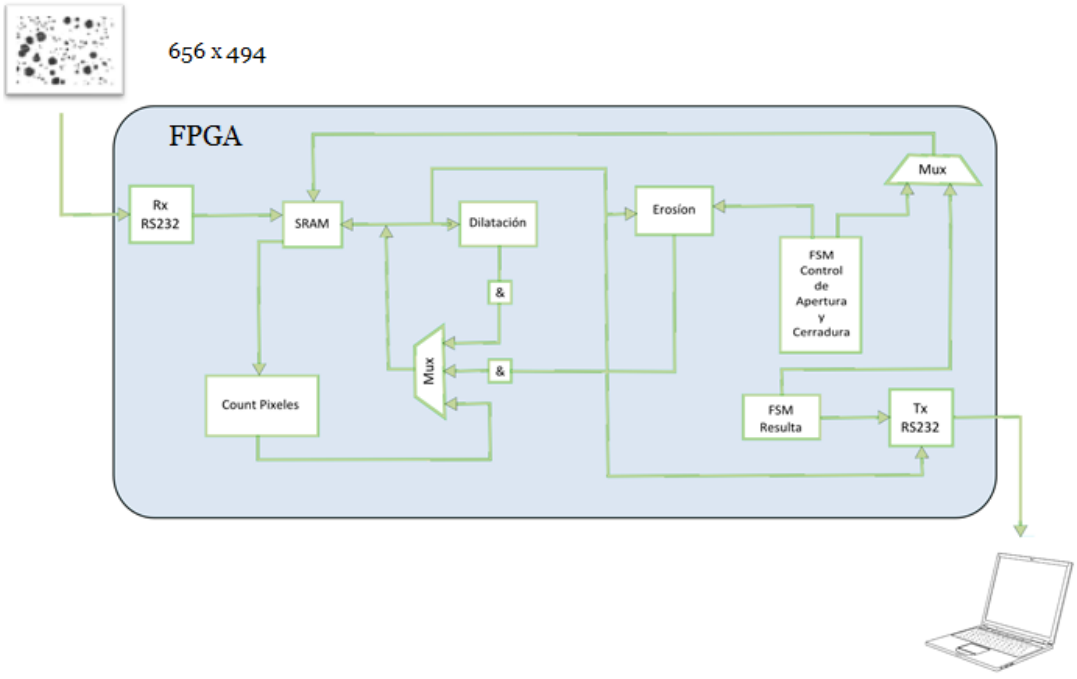
En este trabajo se desarrolla un sensor inteligente, para este caso el sensor primario es el dispositivo con el que se realiza la adquisición de la imagen aunque se tiene la versatilidad de poder hacer la adquisición utilizando diferentes protocolos de comunicación, la etapa de procesamiento es obtener el índice de cuantificación espacial entre objetos con ayuda de los operadores morfológicos y el resultado es obtener dicho índice como se observa en la Figura 3-2.



3-2 Metodología a seguir.

Para este trabajo la función del sensor primario es la siguiente, la imagen es enviada por medio de la PC utilizando el protocolo de comunicación RS-232, una vez que se tienen los datos de la imagen en la FPGA estos son procesados utilizando los operadores morfológicos apertura y cerradura y cada vez que se procesa la imagen es necesario contar y almacenar el número de pixeles de los objetos en la imagen; una vez que termina el procesamiento en la FPGA, los datos son enviados a la PC para realizar la comparación de los resultados obtenidos en hardware contra los datos obtenidos por software y así también poder representar el espectro de tamaños en una gráfica y calcular el índice de cuantificación.

De tal manera que el sensor inteligente queda representado de una manera muy general como se observa en la Figura 3-3, ya con los módulos desarrollados e implementados en el FPGA Altera.

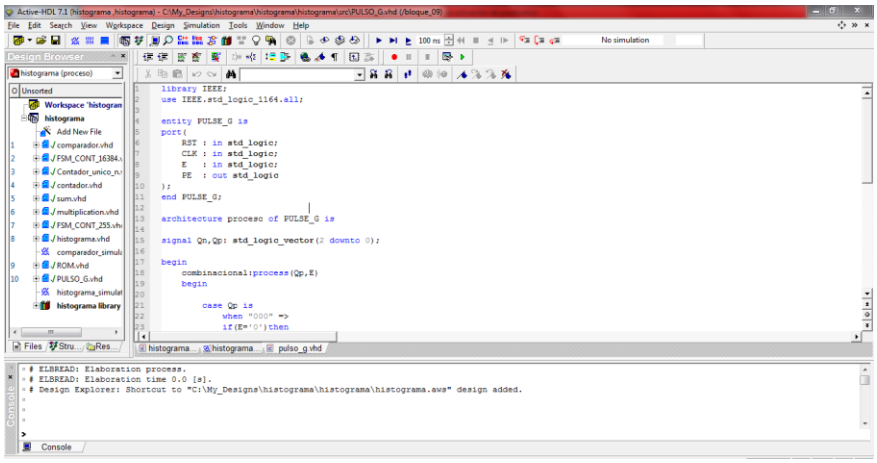


3-3 Diagrama general de la implementación del sensor inteligente en el FPGA.

Haciendo la comparación de la Figura 3-2 con la Figura 3-3, se tiene los siguiente, el sensor primario es el módulo de comunicación serial RS232, ya que gracias a este se obtiene la imagen metalográfica y esta a su vez es almacenada en la SRAM, aunque como se mencionó anteriormente puede ser utilizado cualquier protocolo de comunicación para obtener la imagen ya sea USB, Ethernet, comunicación inalámbrica, etc., una vez almacenada la imagen en la SRAM sigue el módulo de procesamiento, en este caso son los operados morfológicos de erosión y dilatación controlados por una FSM para hacer el cálculo de las aperturas y cerraduras deseadas y finalizado el cálculo de las imágenes procesadas contar el número de pixeles y con ayuda de estos valores una vez que sean enviados a la PC se calcule el espectro de tamaño de las partículas y el índice de cuantificación espacial y así también poder visualizar el resultado gráficamente en la PC.

3.1 Descripciones digitales.

Una vez que se ha comprendido los algoritmos que se utilizarán es necesario desarrollar las descripciones digitales de los mismos y así implementarlos en el FPGA, para realizar dicha acción se hace uso del software Active-HDL (Figura 3-4), el cual permite desarrollar y simular las descripciones digitales.



3-4 Software Active-HDL.

Una descripción digital es un algoritmo o proceso implementado con ayuda de compuertas lógicas y otros dispositivos electrónicos como circuitos integrados, conectores (DB9, Ethernet, Bluetooth), displays, etc..

Desarrollar las descripciones digitales de los operadores morfológicos erosión y dilatación, son parte fundamental del desarrollo de este trabajo ya que

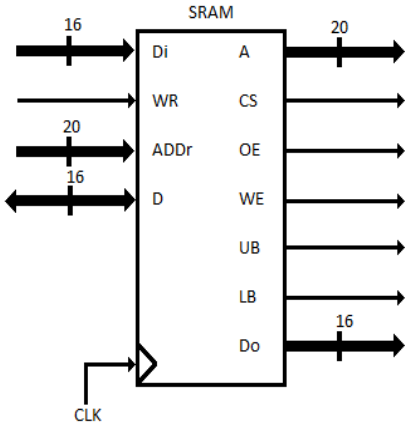
gracias a la combinación de estos se obtienen los operadores morfológicos apertura y cerradura, los cuales como se vio en la fundamentación teórica permiten calcular los tamaños y distancias de los nódulos y con ello obtener el índice de cuantificación espacial a calcular.

Una gran ventaja de realizar los módulos (descripciones digitales) tanto de erosión como dilatación, es que al ser un core de nuestra propiedad (I.P. Core, *Intelectual Property Core*) es posible acoplarlos a otros proyectos, por ello, es importante realizar los módulos de las maneras más eficiente posible para así poder tener control sobre ellos. Por ello estos módulos cuentan con una señal de *start* o Inicio y con la señal de Fin.

3.1.1 Módulo para la SRAM de la Altera DE2-115.

Para el desarrollo de este trabajo se debe de manipular la memoria estática con la que cuenta la FPGA ALTERA DE2-115, para ello es necesario realizar el controlador o driver que permita almacena y leer datos en la SRAM (*Static Random Access Memory*, Memoria estática de acceso aleatorio), los tiempos de lectura y escritura con los que funciona la memoria IS61WV102416BLL-10TL son de 10 ns y 20 ns respectivamente, esta memoria cuenta con una capacidad de almacenamiento de 4 MB, las palabras o datos son de 16 bits y las direcciones son de 20 bits.

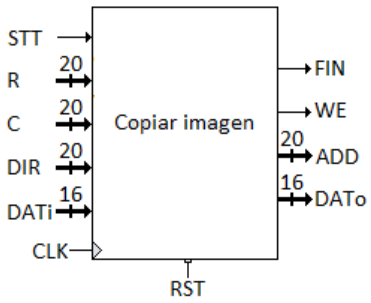
En la Figura 3-5 se muestran las entradas y salidas con las que cuenta dicho dispositivo electrónico.



3-5 Diagrama de la SRAM

3.1.1 Módulo copiar imagen.

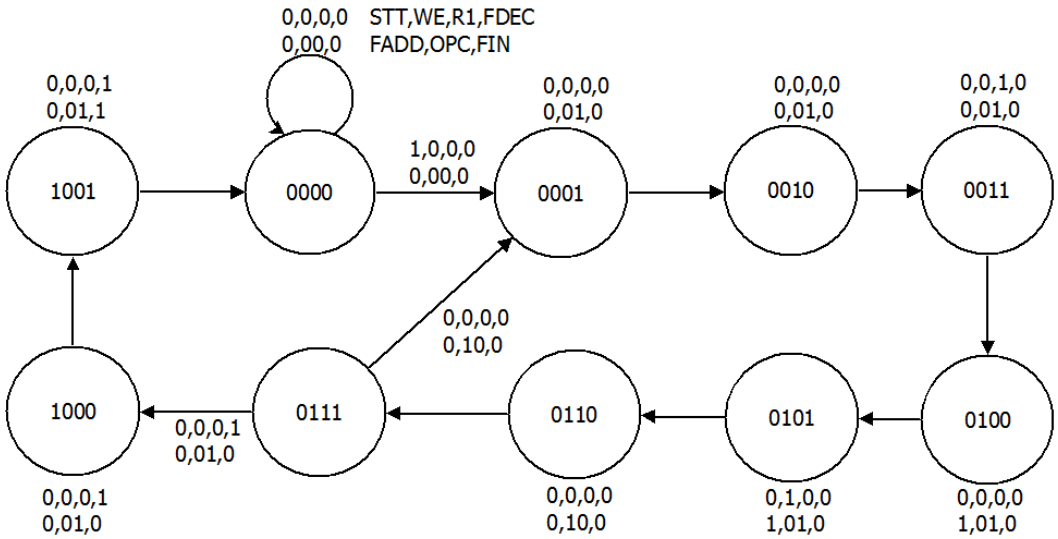
Debido a que los algoritmos (procesamientos de imágenes) que se utilizan para calcular el índice de cuantificación espacial modifican la imagen a procesar se debe de hacer una copia de la misma para poder trabajar sobre ésta y disponer siempre de la imagen original, para ello es desarrollado este módulo el cual permite hacer la copia de la imagen.



3-6 Módulo que permite hacer copia imagen.

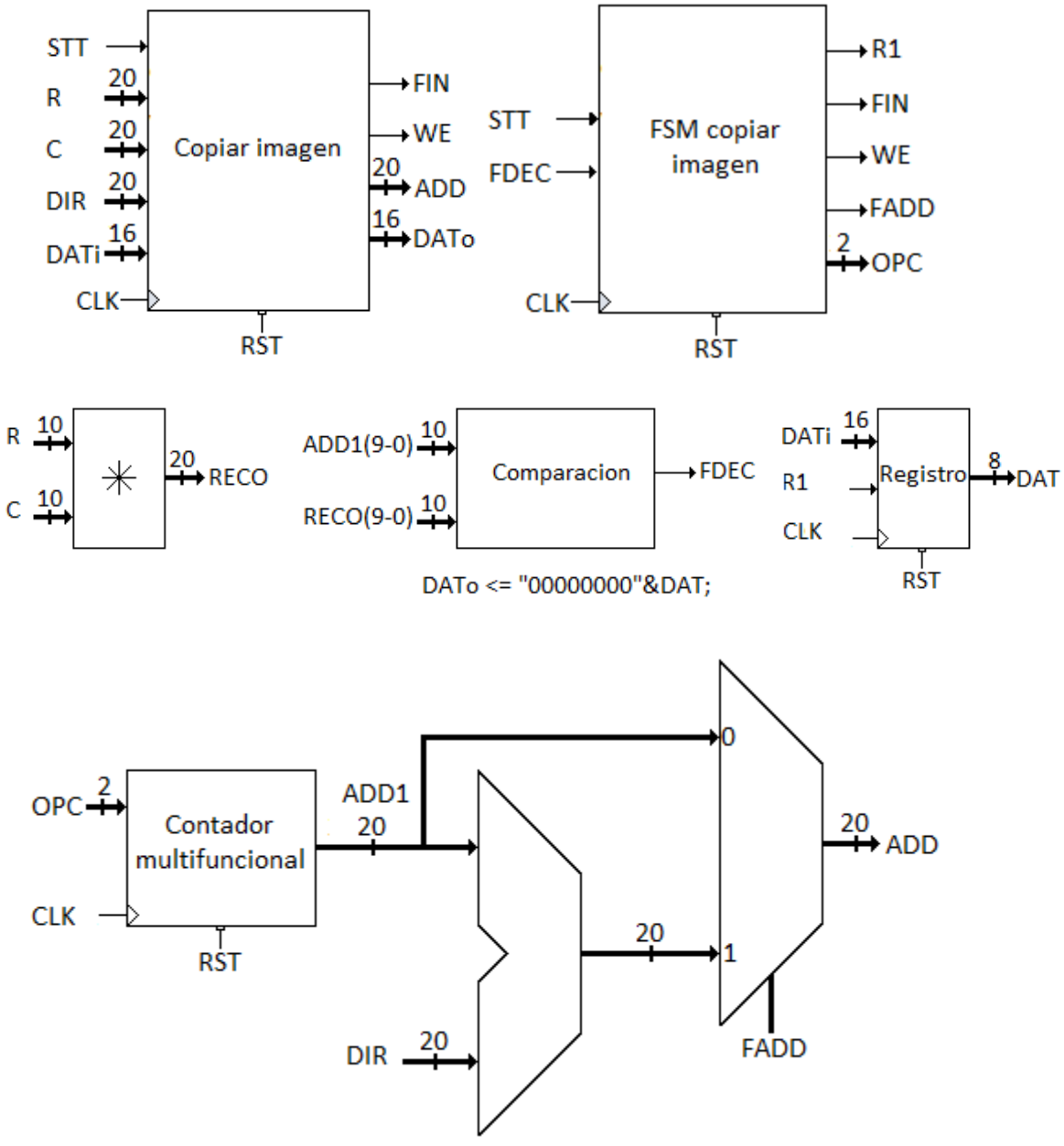
Para iniciar a trabajar este módulo es necesario recibir una señal de inicio (STT), el número de renglones (R) y columnas (C) de la imagen a copiar, la dirección donde se desea iniciar a hacer la copia de la imagen (DIR) y los datos que se desea copiar (DATi); como salidas tiene la señal de dirección (ADD), la señal de escritura/lectura (WE), la señal del dato de salida a escribir en la SRAM (DATo) y la señal de fin de proceso (FIN).

Este módulo posee internamente una FSM (*Finite State Machine*, Máquina de estados finitos) para dar control a su funcionamiento, en la Figura 3-7 se muestra la FSM desarrollada.



3-7 FSM copiar imagen.

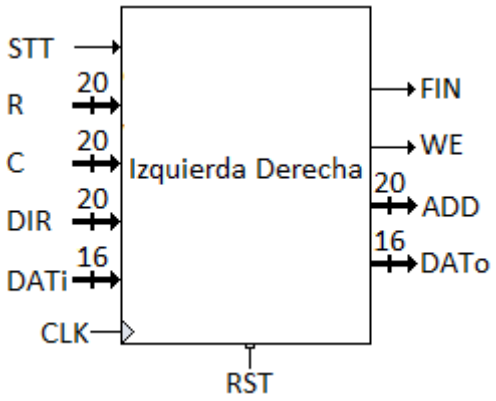
En la Figura 3-8 se muestra la estructura digital desarrollada para este módulo de copiar imagen, es utilizado un contador multifuncional y su funcionamiento es el siguiente si la señal OPC es igual a “00” pone la cuenta en ceros, si es “01” mantiene el dato, si es “10” incrementa en uno y si es “11” el decremento es en uno.



3-8 Estructura Digital implementada.

3.1.2 Módulo izquierda derecha.

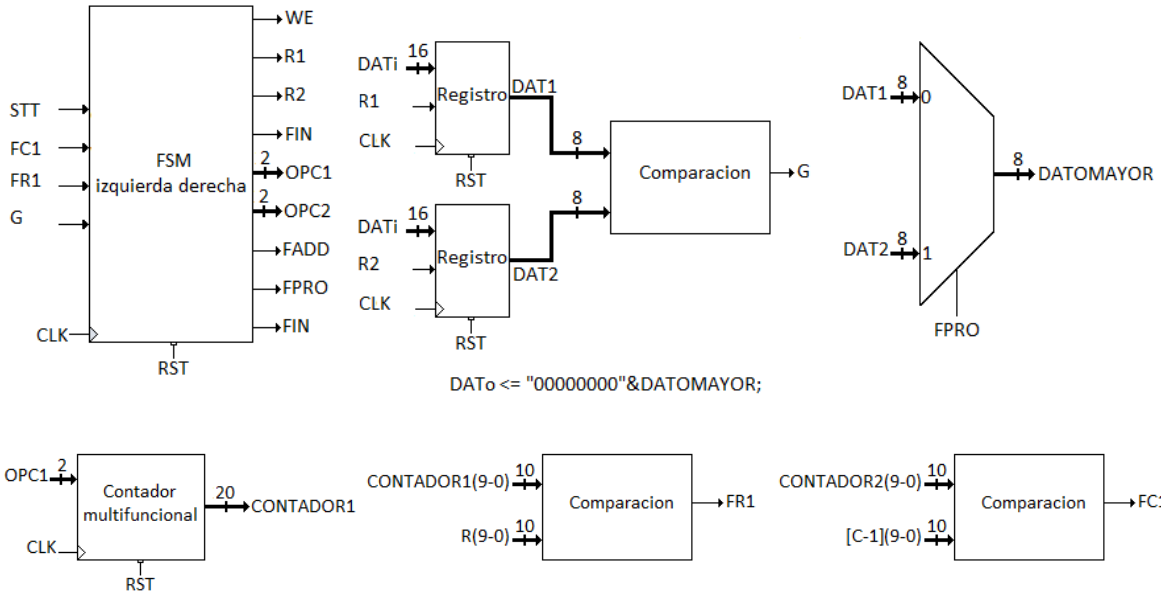
Este módulo realiza el procesamiento llamado barrido izquierda derecha visto en el capítulo 2, en la Figura 3-9 se muestra el módulo con sus respectivas señales de entrada y salida.

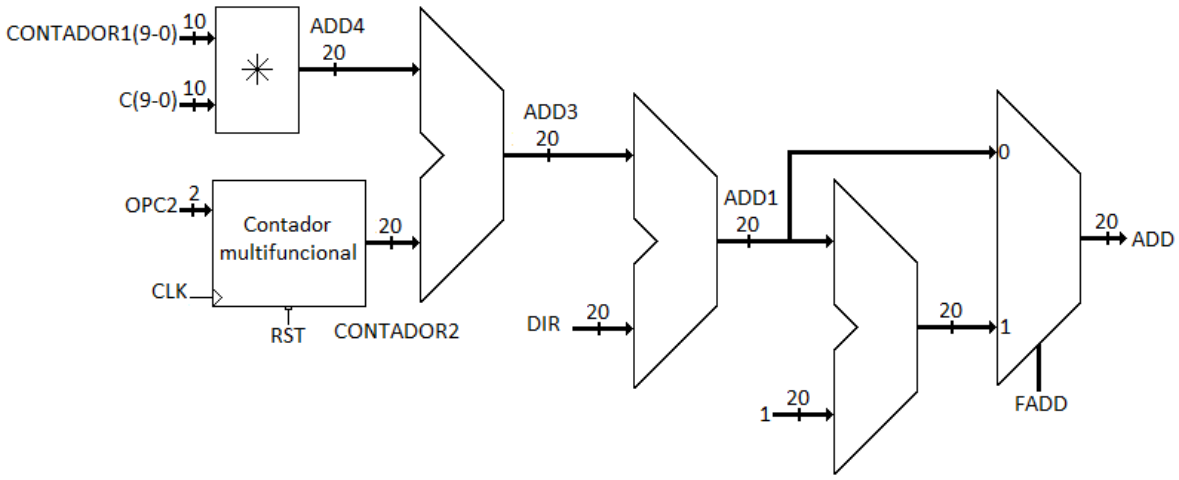


3-9 Módulo izquierda-derecha.

Como se observa en la Figura 3-9, el módulo posee las mismas señales que el módulo de copiar imagen, señal de inicio (STT), renglones(R), columnas(C), dirección donde se desea empezar el procesamiento (DIR), dato de entrada (Dato de salida de la SRAM, DATi), señal de escritura/lectura(WE), dirección(ADD), dato de salida(DATo) y señal de finalización (FIN).

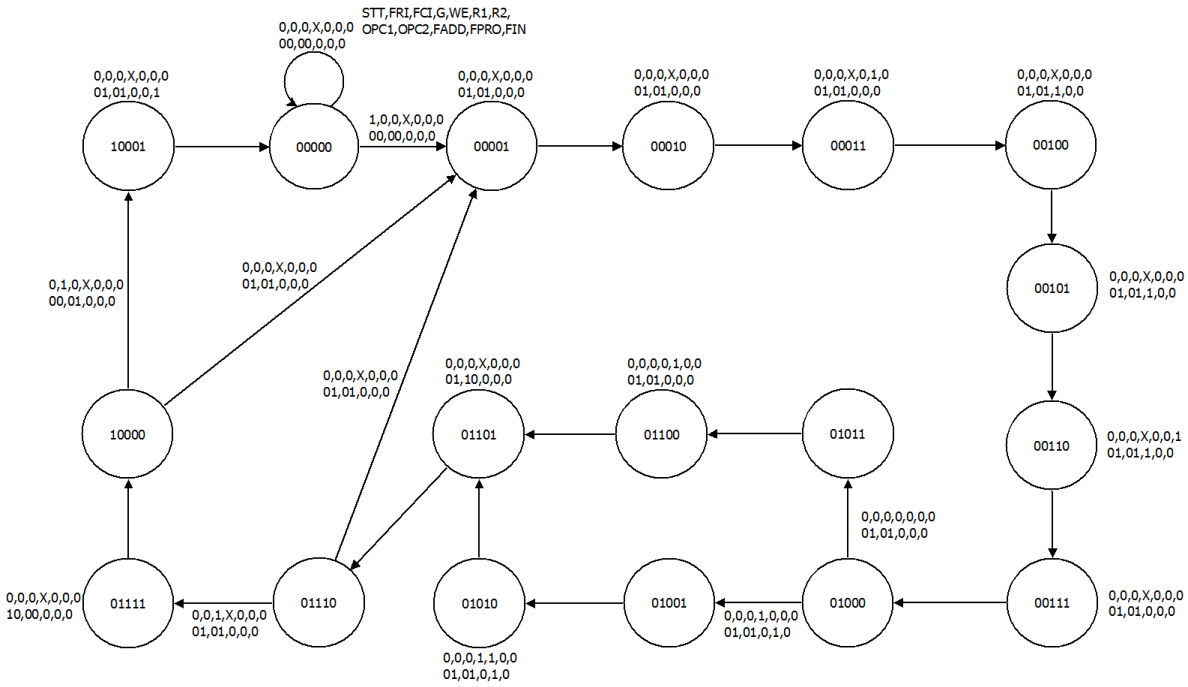
La estructura digital desarrollada para realizar esta tarea se muestra en la Figura 3-10, es importante señalar que dependiendo del operador morfológico a utilizar se debe de considerar el valor de la señal G (Ver apéndices).





3-10 Estructura digital para módulo izquierda derecha.

La FSM interna desarrollada que posee este módulo se muestra en la Figura 3-11.



3-11 FSM izquierda derecha.

3.1.3 Módulo arriba abajo.

Este módulo tiene las mismas señales de entrada y salida que el módulo de izquierda derecha

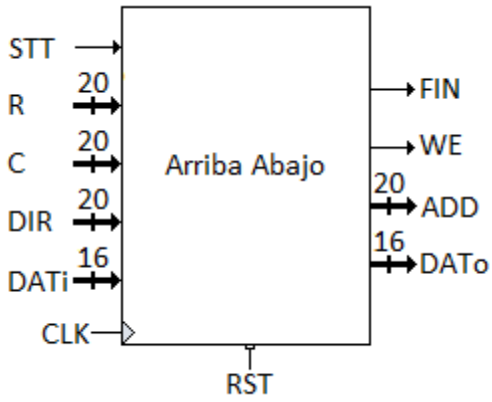
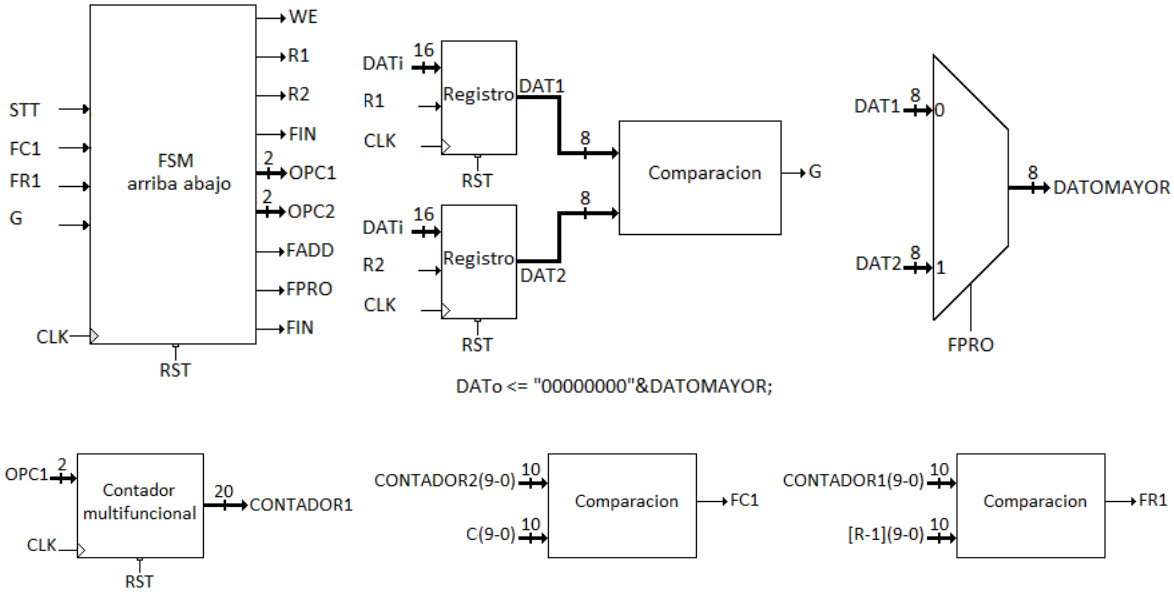
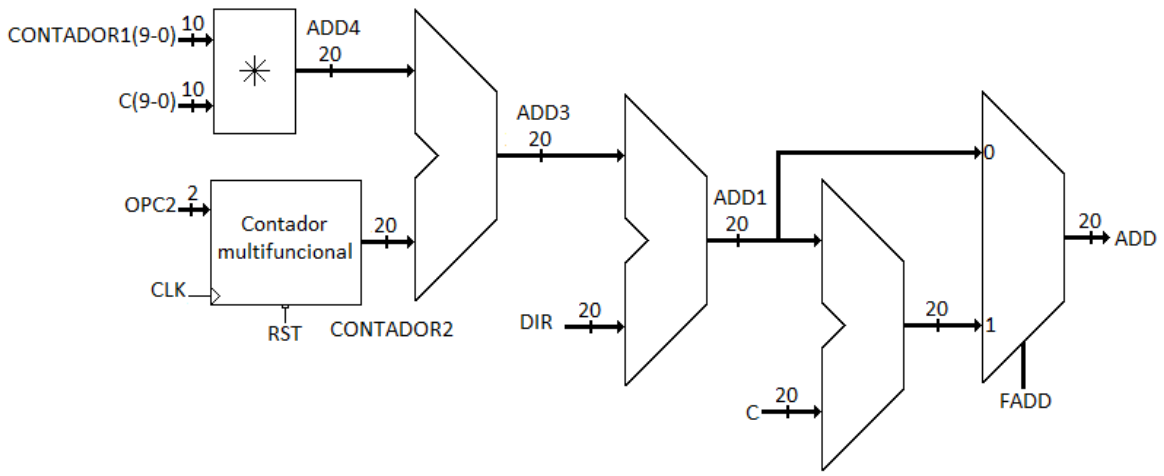


Figura 3-12 Módulo arriba abajo.

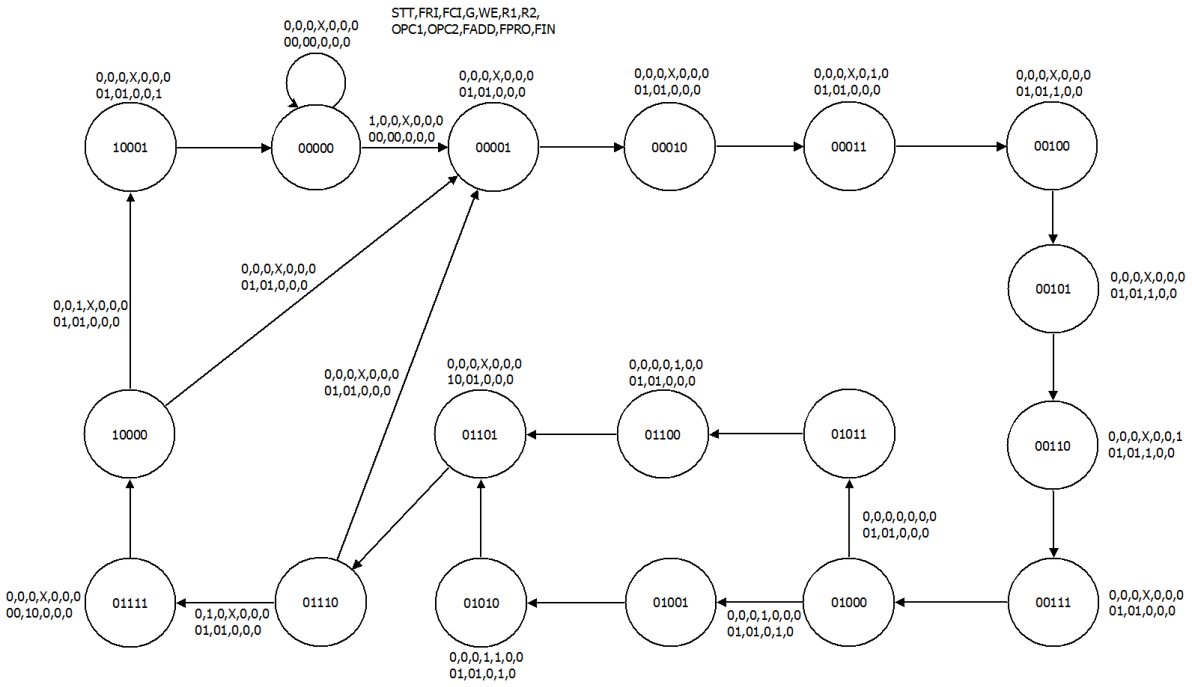
La estructura digital desarrollada para este módulo se muestra en la Figura 3-13.





3-13 Estructuras digitales para módulo arriba abajo.

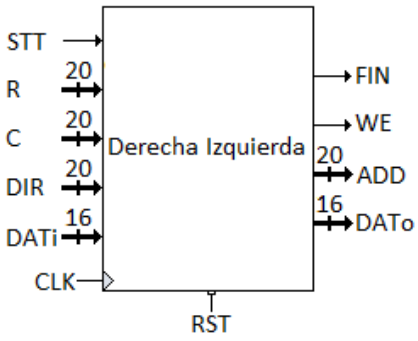
En la Figura 3-14 se muestra la FSM desarrollada para este módulo.



3-14 FSM arriba abajo.

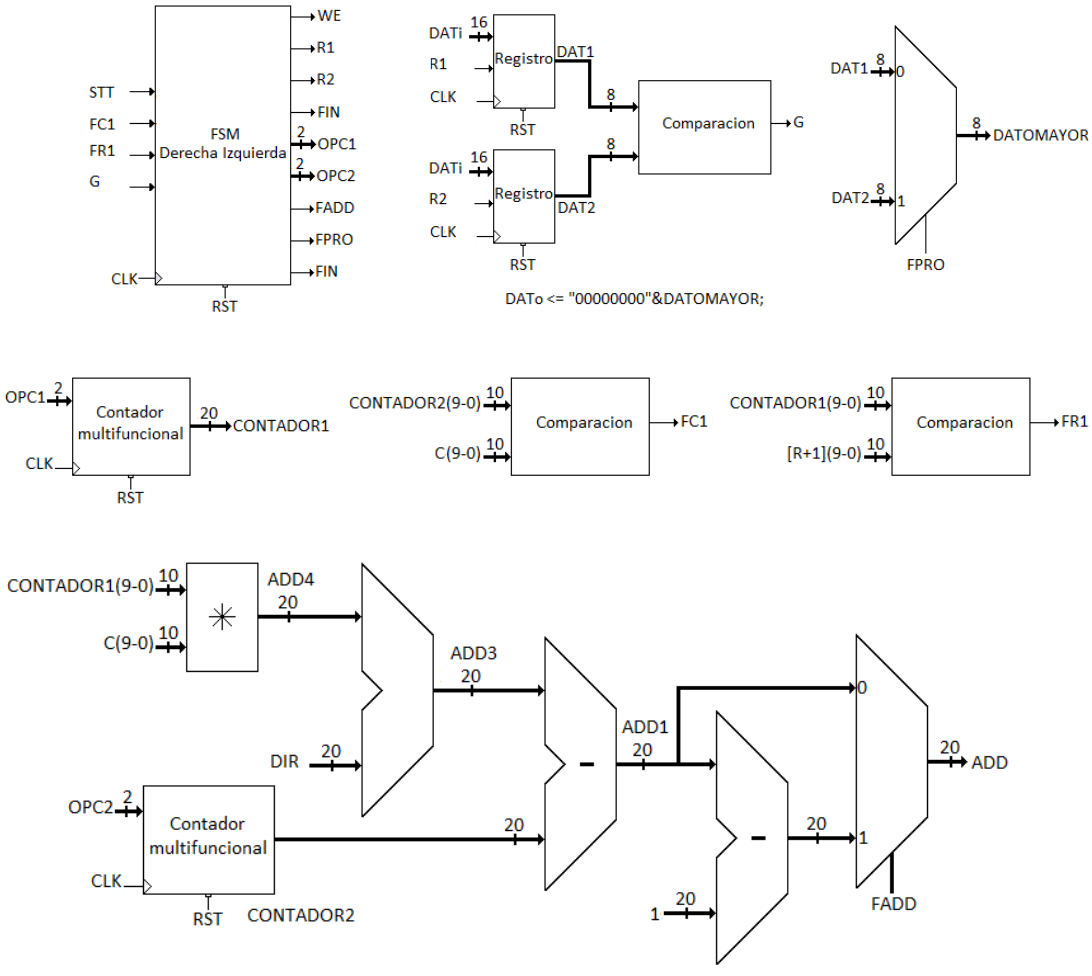
3.1.4 Módulo derecha izquierda.

En la Figura 3-15 se muestra el módulo derecha izquierda.



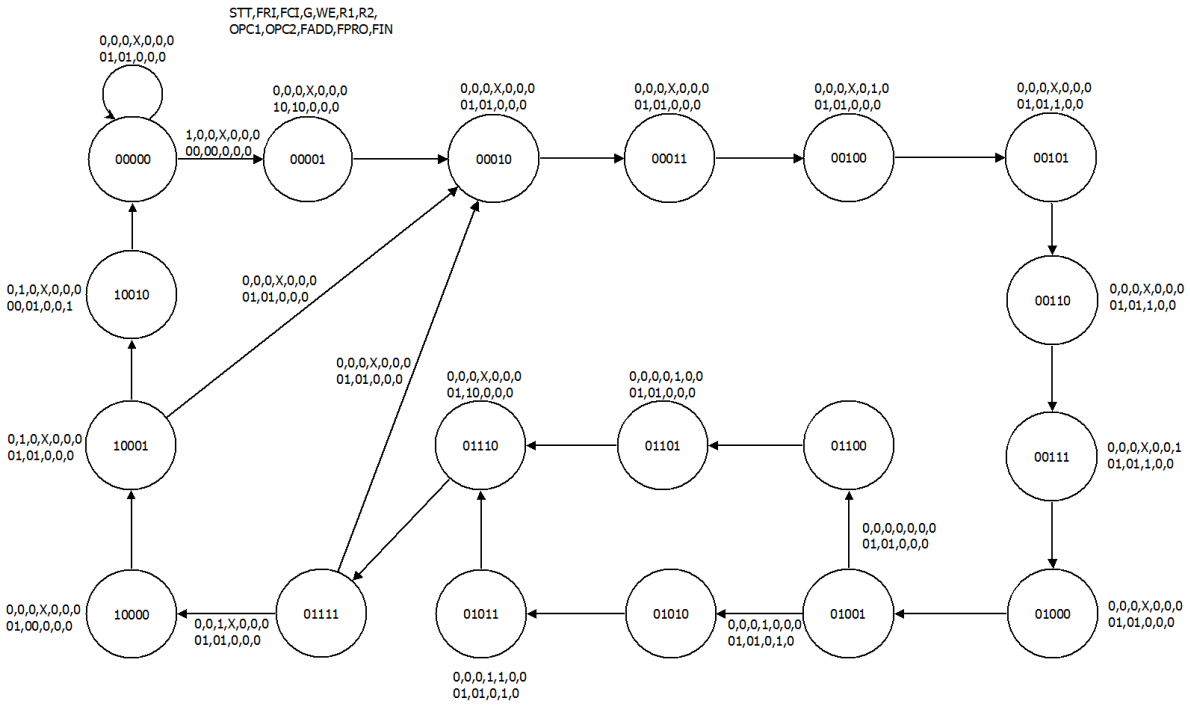
3-15 Módulo derecha izquierda.

La estructura digital desarrollada para este módulo se muestra en la Figura 3-16.



3-16 Estructuras digitales para el módulo derecha izquierda.

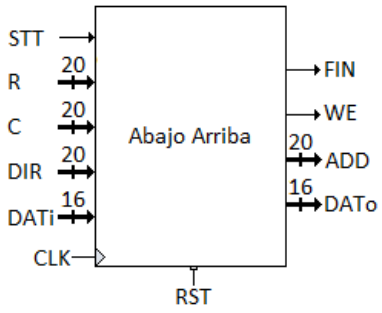
En la Figura 3-17 se muestra la FSM desarrollada para este módulo.



3-17 FSM derecha izquierda.

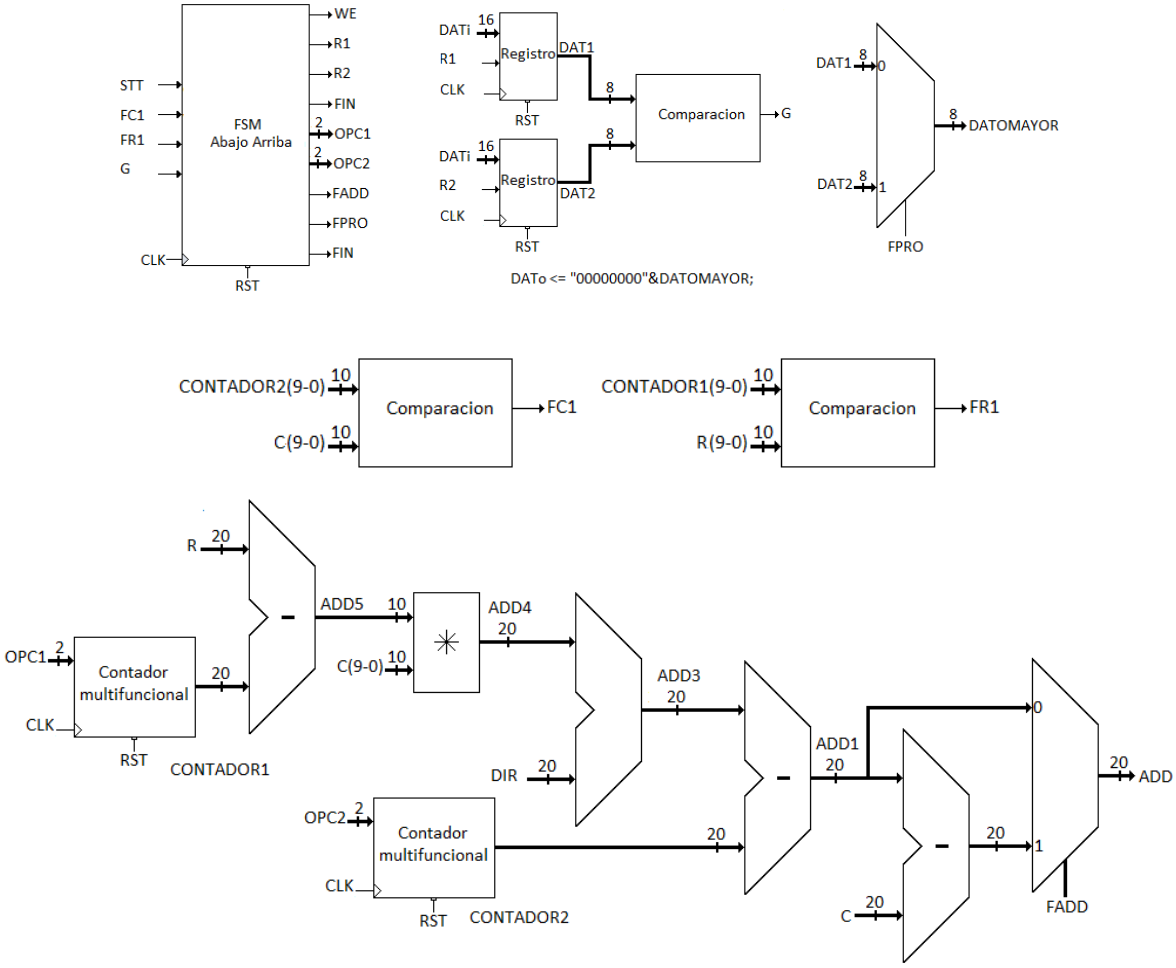
3.1.5 Módulo abajo arriba.

En la Figura 3-18 se muestra el módulo abajo arriba.



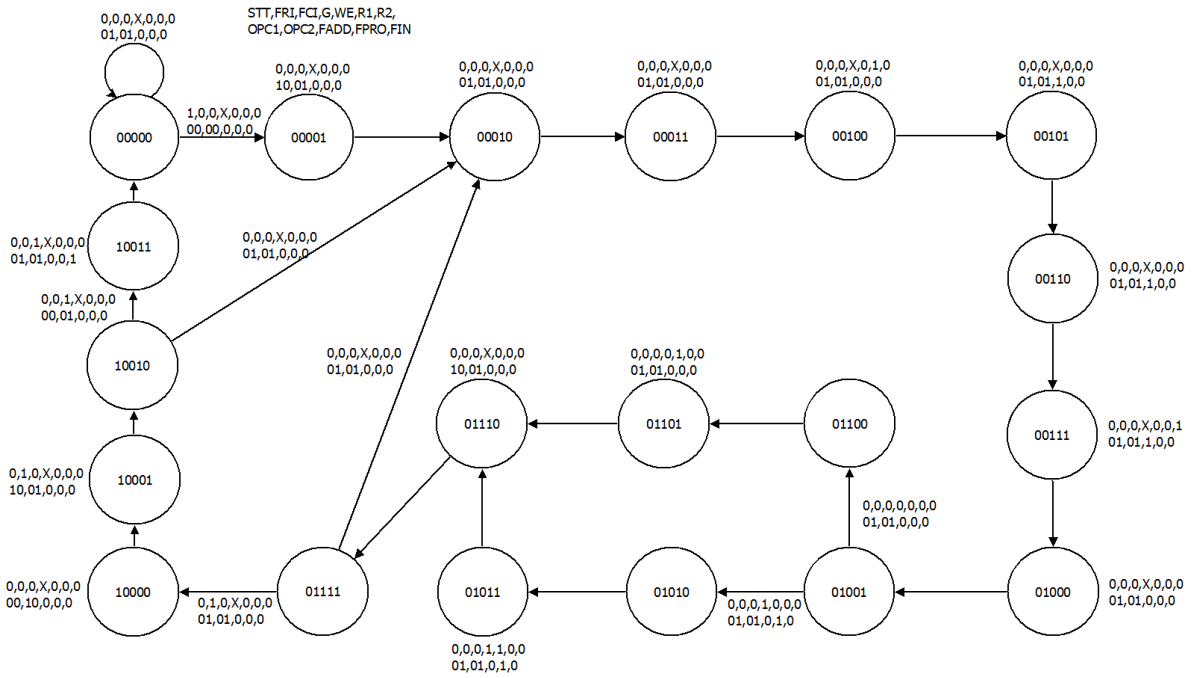
3-18 Módulo abajo arriba.

La estructura digital desarrollada para este módulo se muestra en la Figura 3-19.



3-19 Estructura digitales para el módulo abajo arriba.

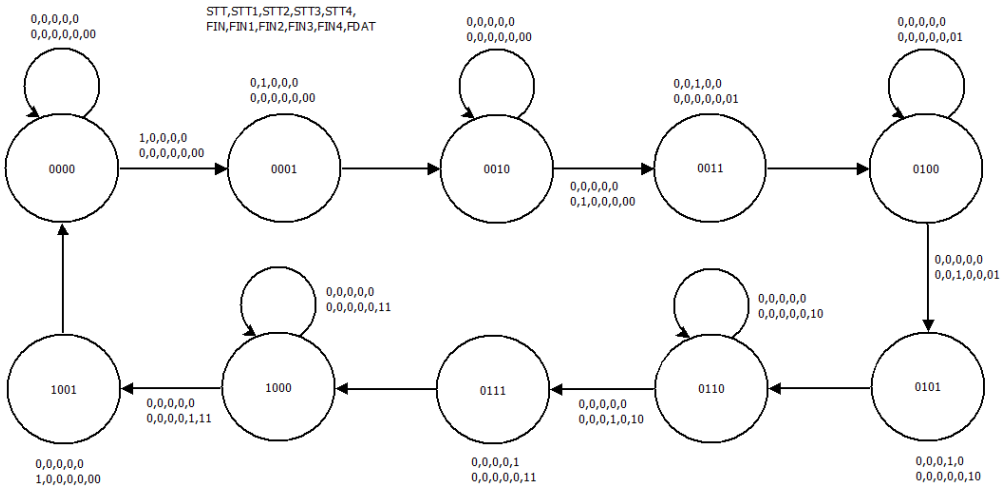
En la Figura 3-20 se muestra la FSM desarrollada para este módulo.



3-20 FSM abajo arriba.

3.1.6 Módulos erosión y dilatación.

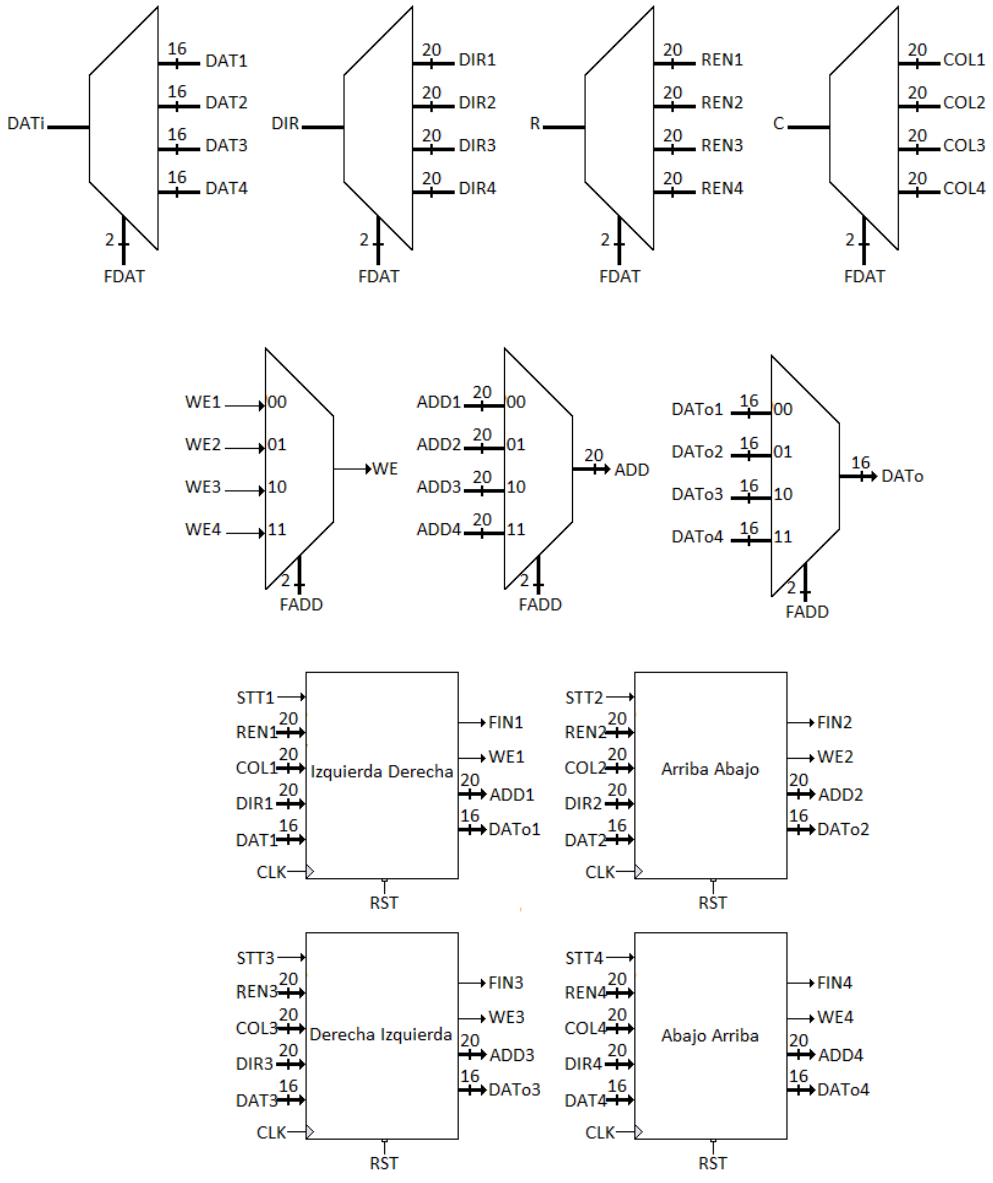
Una vez que se tienen los cuatro módulos anteriores, lo que prosigue es hacer una FSM que nos permita controlar el orden en que se desea llevar a cabo las rutinas de los módulos. Como se observó anteriormente para obtener el módulo de los operadores morfológicos erosión y dilatación es necesario ejecutarlos en el siguiente orden: izquierda derecha, arriba abajo, derecha izquierda y abajo arriba.



3-21 FSM control 4.

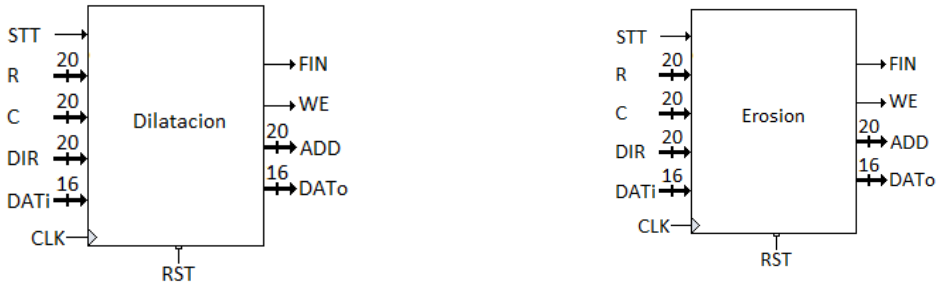
La FSM desarrollada para controlar dicho proceso se muestra en la Figura 3-21, dicha FSM cuenta con una señal de inicio (STT), una señal que permite controlar el flujo de datos a los módulos (FDAT), las señales de inicio y fin para cada uno de los cuatro módulos y la señal de fin del proceso del módulo (FIN).

La señal FDAT permite controlar las señales a la SRAM con ayuda de demultiplexores 1-4 y multiplexores 4-1, en la Figura 3-22 se muestran los demultiplexores y multiplexores utilizados.



3-22 Demultiplexores y multiplexores para la distribución de las señales a los módulos.

De tal manera que el módulo de erosión y dilatación quedan como se observa en la Figura 3-23.

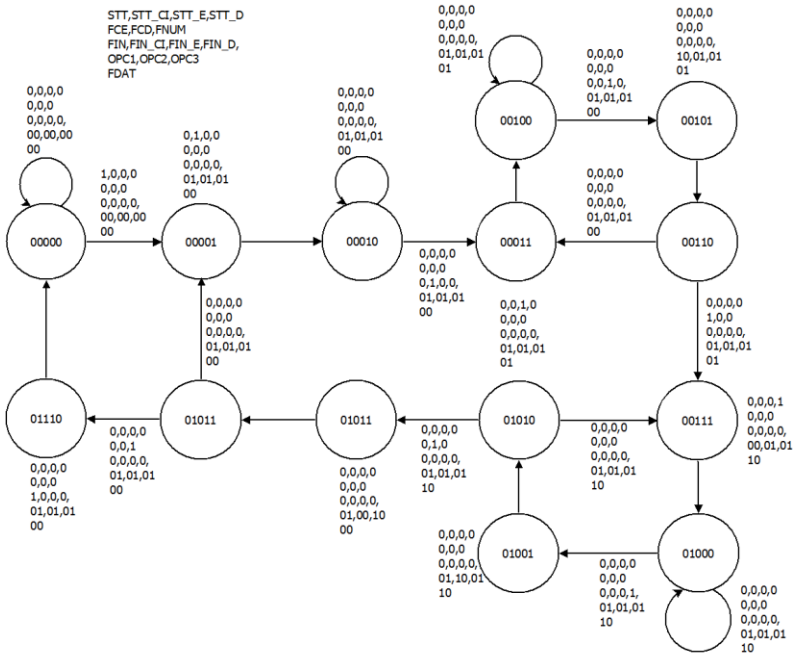


3-23 a) Módulos Dilatación, b) Módulo Erosión.

3.1.7 Módulos apertura y cerraduras.

Para obtener el resultado del operador morfológico apertura o cerradura, es necesario aplicar *n* número de veces el operador erosión seguido de *n* dilataciones para obtener la apertura *n*, en caso de que se desee la cerradura, primero se aplica *n* dilataciones seguida de *n* erosiones, donde *n* es el valor de la apertura o cerradura que se desee calcular.

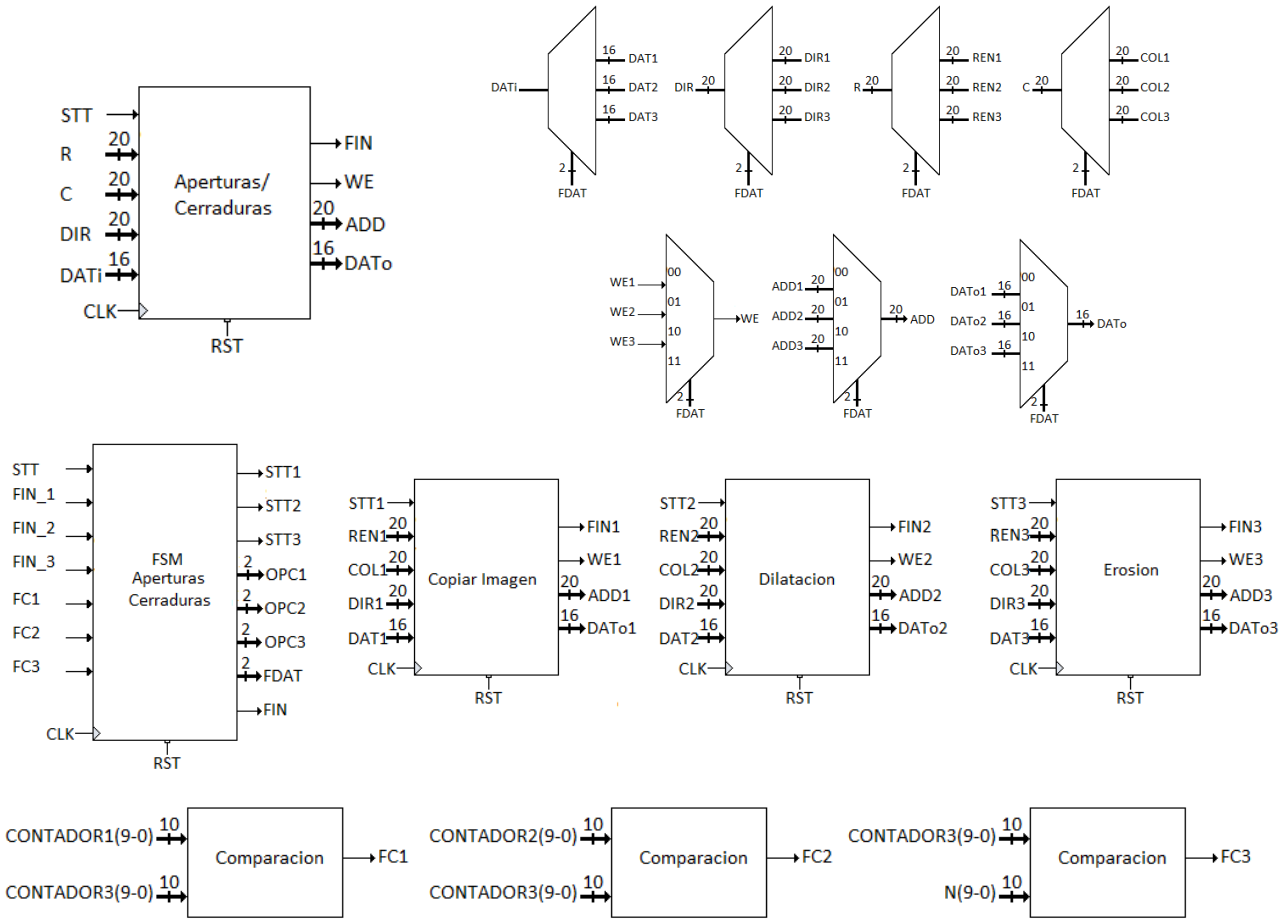
Para lograr obtener las aperturas o cerraduras deseadas es necesario desarrollar una FSM que permita control la rutina de ejecución de los módulos de erosión y dilatación, la FSM desarrollada se muestra en la Figura 3-24.



3-24 FSM para calcular aperturas o cerraduras.

Cabe mencionar que la FSM puede ser utilizada tanto para calcular *n* aperturas o cerradura.

En la Figura 3-25 se muestran las estructuras digitales desarrolladas para poder calcular las aperturas o cerraduras.



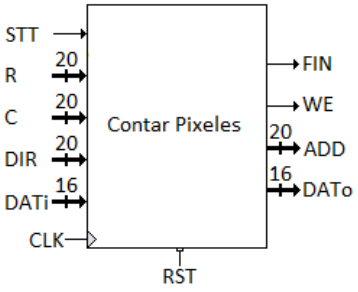
3-25 Descripciones digitales para módulo de aperturas o cerraduras.

Las señales FC1, FC2 y FC3 son las mismas señales FCE (Bandera del contador erosiones), FCD (Bandera del contador dilataciones) y FNUM (Bandera de cuantas aperturas o cerraduras se desean calcular, *N*) representadas en al FSM siguiente, estas banderas se activan una vez que se han completado el número de iteraciones correspondientes.

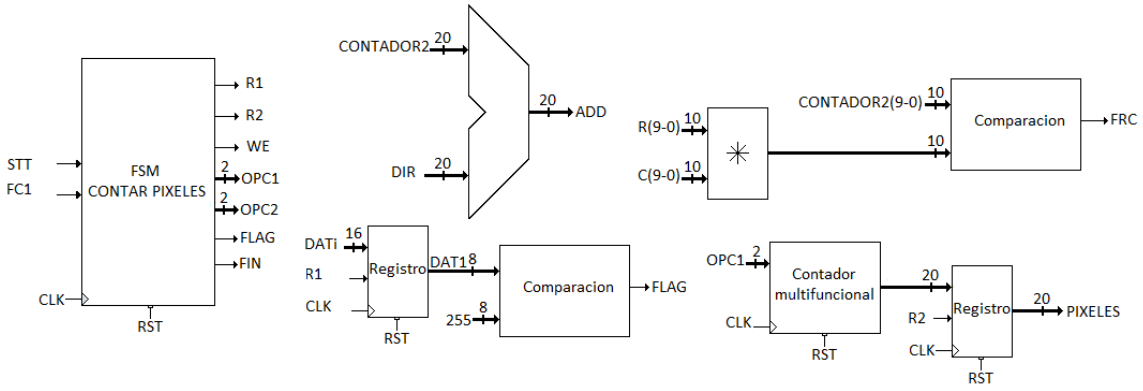
3.1.8 Módulo contar pixeles.

Una vez que son calculadas las aperturas o cerraduras es necesario conocer el número de pixeles de interés en la imagen procesada, para que una vez que se tengan estos datos se pueda aplicar la granulometría y calcular el espectro de tamaños.

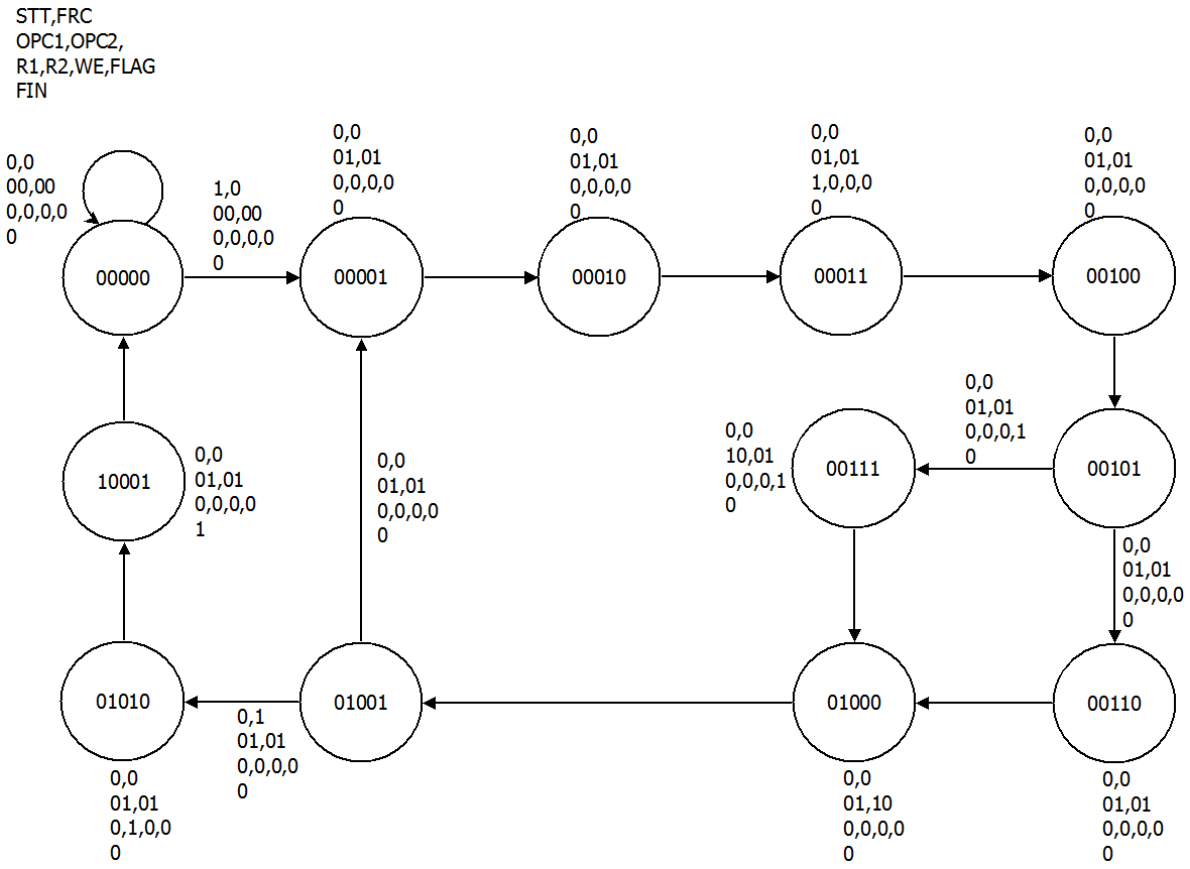
Es por ello que se desarrolló este módulo que realiza el barrido a la imagen procesada y cuenta el número de pixeles de interés. En la Figura 3-26 se muestra el módulo de contar pixeles, en la Figura 3-26 las estructuras digitales desarrolladas y en la Figura 3-28 la FSM utilizada para lograr hacer dicha tarea.



3-26 Módulo contar pixeles.



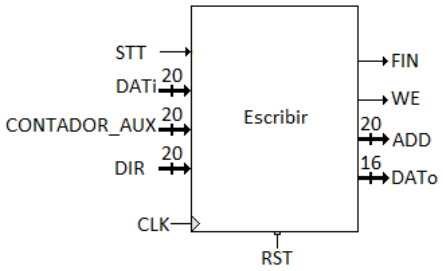
3-27 Estructuras digitales para módulo contar pixeles.



3-28 FSM módulo contar pixeles.

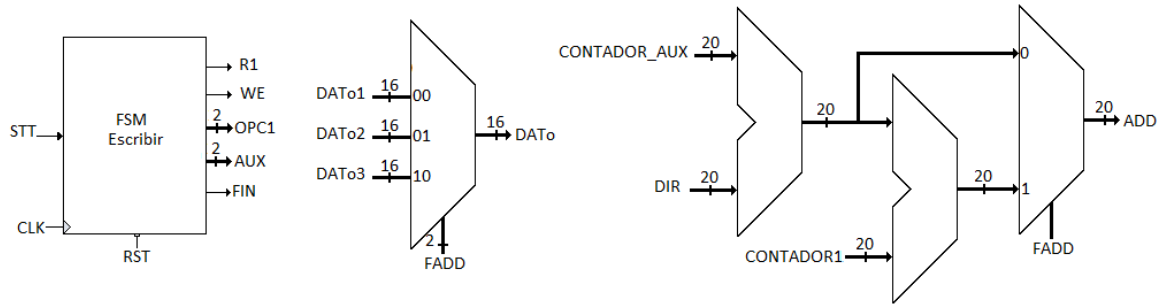
3.1.9 Módulo escribir cuenta pixeles.

Una vez que se tiene el número de pixeles de interés, es necesario almacenar dichos datos en la SRAM, es por ello que se desarrolló este módulo el cual permite almacenar datos para imágenes de tamaño máximo de 640 x480.

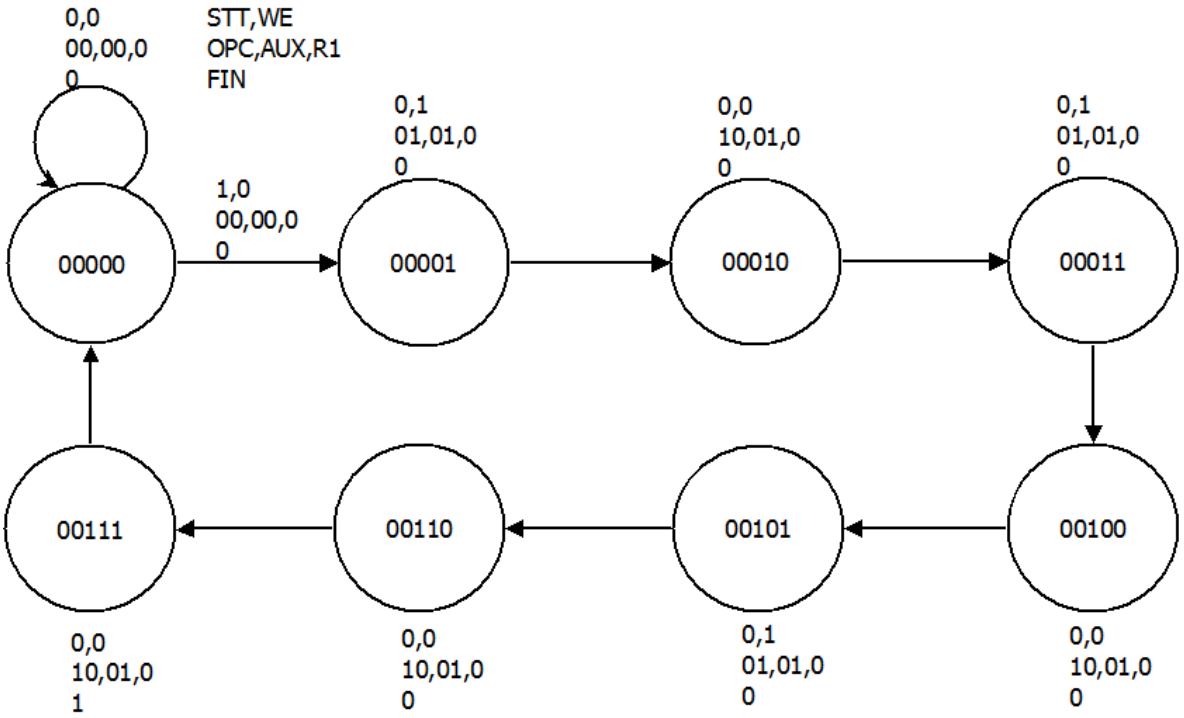


3-29 Módulo escribir cuenta pixeles.

La estructura digital desarrollada para este módulo se muestra en la Figura 3-30 y su respectiva máquina de estados en la Figura 3-31.



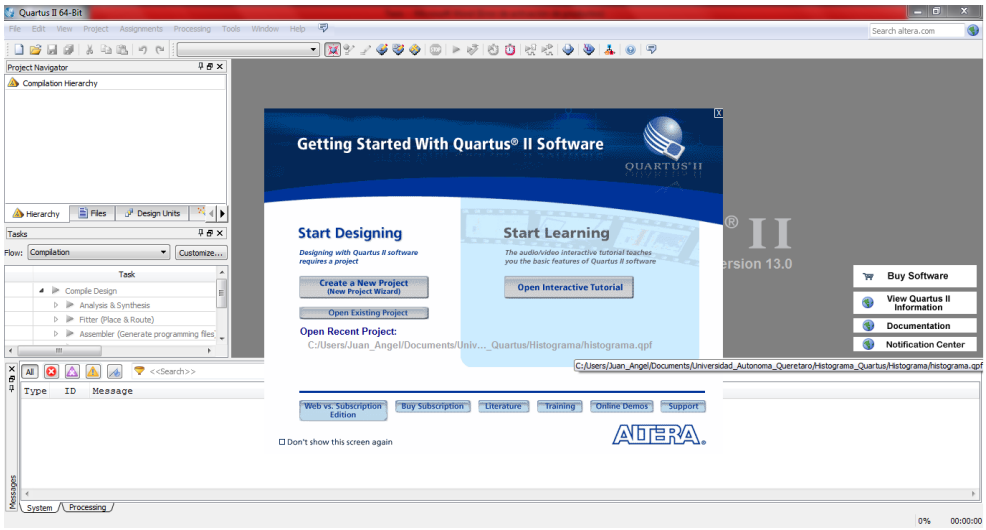
3-30 Estructura digital.



3-31 FSM escribir cuenta de pixeles.

3.2 Quartus II

El software utilizado para implementar las descripciones digitales desarrolladas en Active-HDL en la FPGA Altera es el software de Quartus II, en este software es necesario definir las entradas y salidas de nuestras descripciones digitales para que las entradas y salidas sean conectadas a los dispositivos electrónicos utilizados, en este caso la SRAM y el puerto de comunicación serial SR232, en la Figura 3-32 se muestra la interfaz del software.



3-32. Interfaz del software Quartus II.

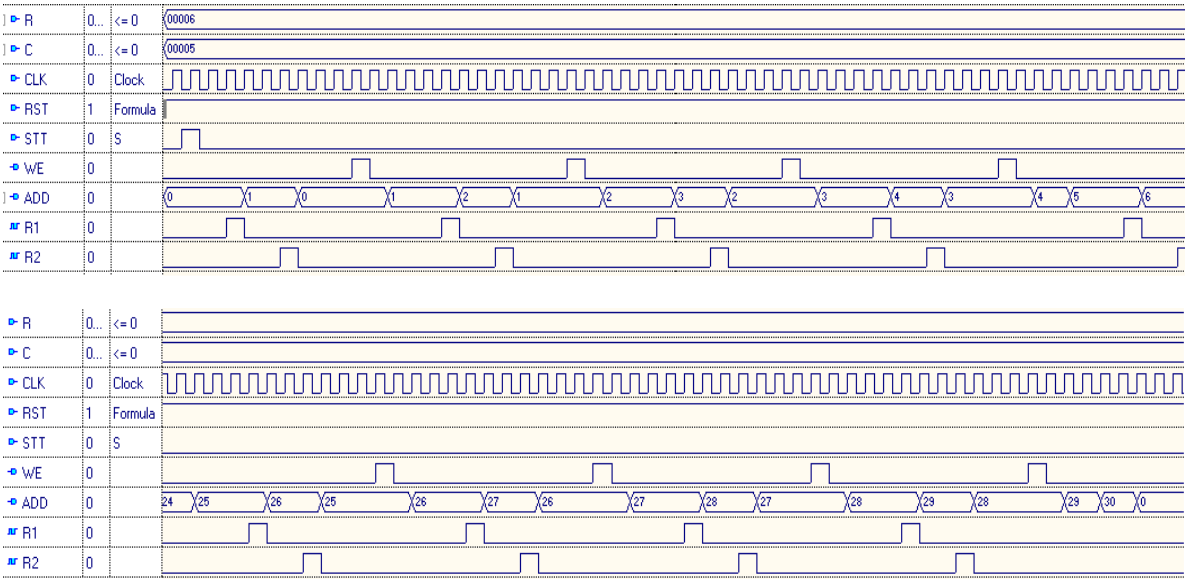
Capítulo 4.
Experimentación y resultados

En este capítulo se presentan las simulaciones y pruebas realizadas a las descripciones digitales desarrolladas e implementadas en hardware, así también se presenta los resultados en software del análisis del espectro de tamaño de los datos calculados en el FPGA.

4.1 Simulaciones de las descripciones digitales.

Las simulaciones se hicieron para imágenes de seis renglones por cinco columnas, esto con el fin corroborar que efectivamente se obtenía la secuencia de direcciones (ADD) para acceder a los datos de la SRAM, solo se muestran las simulaciones de los módulos más importantes los cuales son los barridos denominados izquierda-derecha, arriba-abajo, derecha-izquierda y abajo-arriba.

4.1.1 Simulación módulo izquierda-derecha.

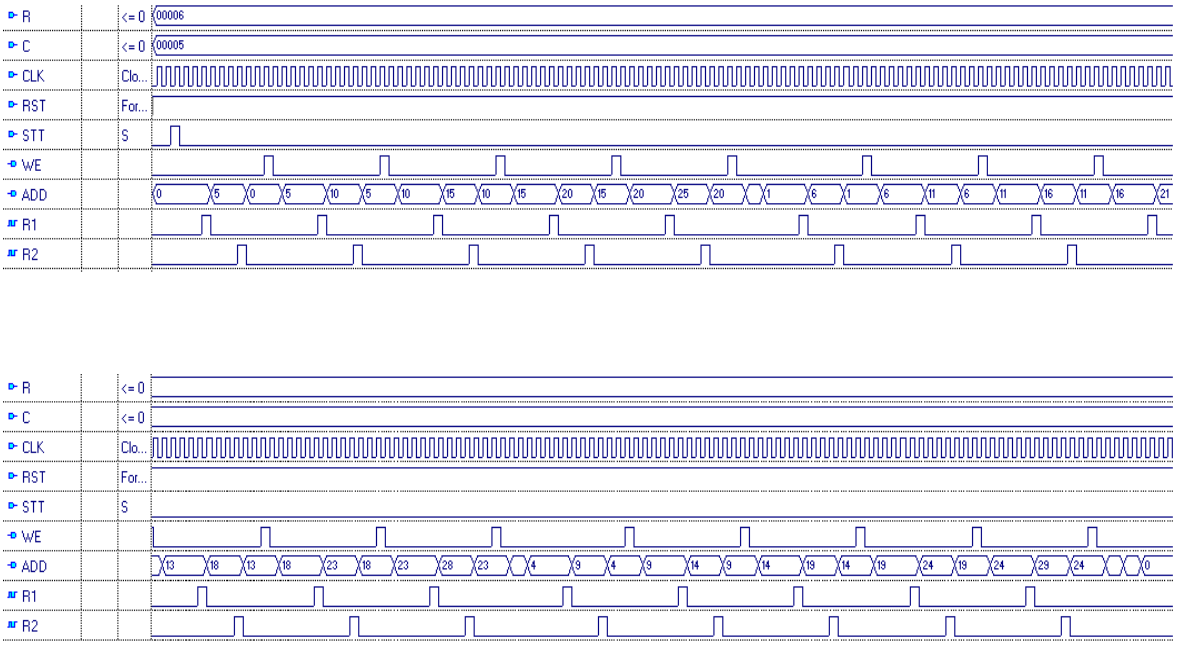


4-1 Simulación módulo izquierda derecha

En la simulación se observan solamente las señales de inicio(STT), reset (RST), señal lectura/escritura (WE) y los registros R1 y R2, donde se almacenaban los datos a leer.

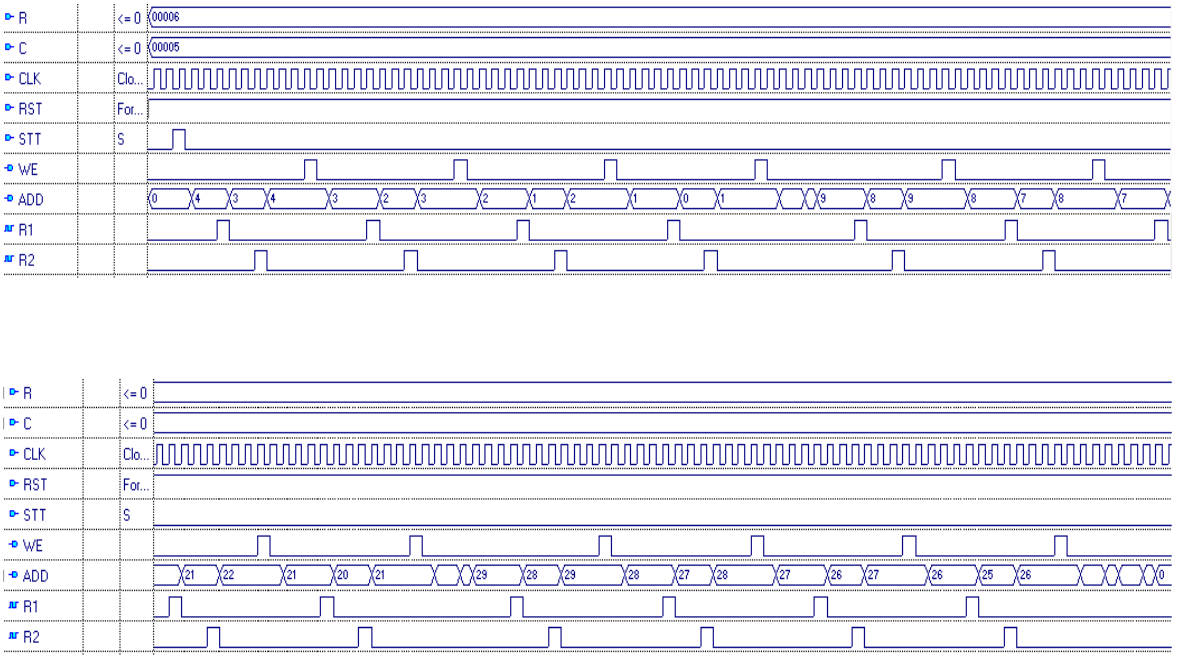
4.1.2 Simulación módulo arriba abajo

De igual manera en esta y las siguientes simulaciones solo se muestran la secuencia de direcciones, las cuales son las señales de mayor prioridad del desarrollo de este trabajo.



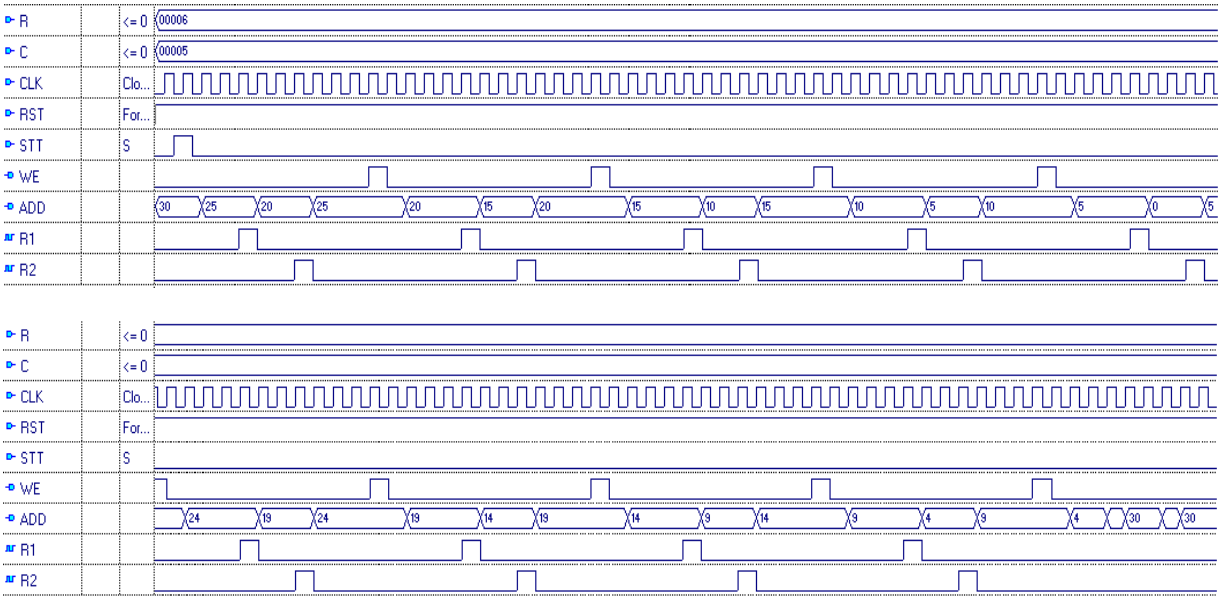
4-2 Simulación módulo arriba abajo

4.1.3 Simulación módulo derecha izquierda



4-3 Simulación módulo derecha izquierda

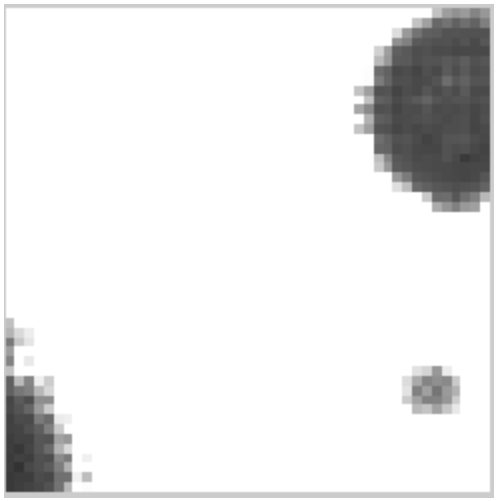
4.1.4 Simulación abajo arriba



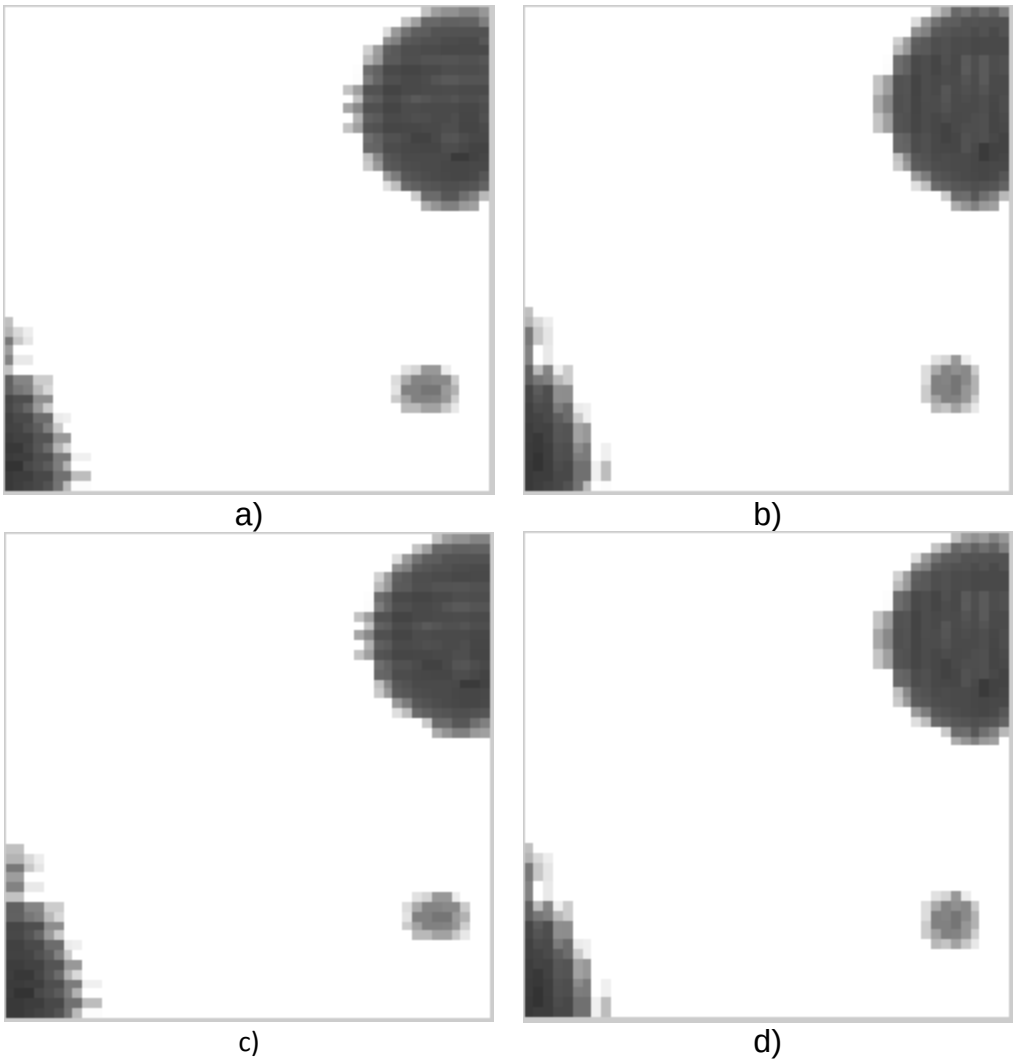
4-4 Simulación módulo abajo arriba

4.1.5 Procesamiento en hardware de los módulos principales.

En la Figura 4-5 se muestra la imagen a niveles de grises de la imágen de muestra de una fundición nodular que se procesó de tamaño 50x50, se eligió este tamaño para observar más claramente cómo se modifica el valor de los pixeles en cada proceso, y en la Figura 4-6 se muestra los resultados obtenidos.



4-5 Imagen de la muestra de una fundición nodular



4-6 Procesamiento: a) izquierda-derecha, b) arriba-abajo, c) derecha-izquierda y d) abajo-arriba.

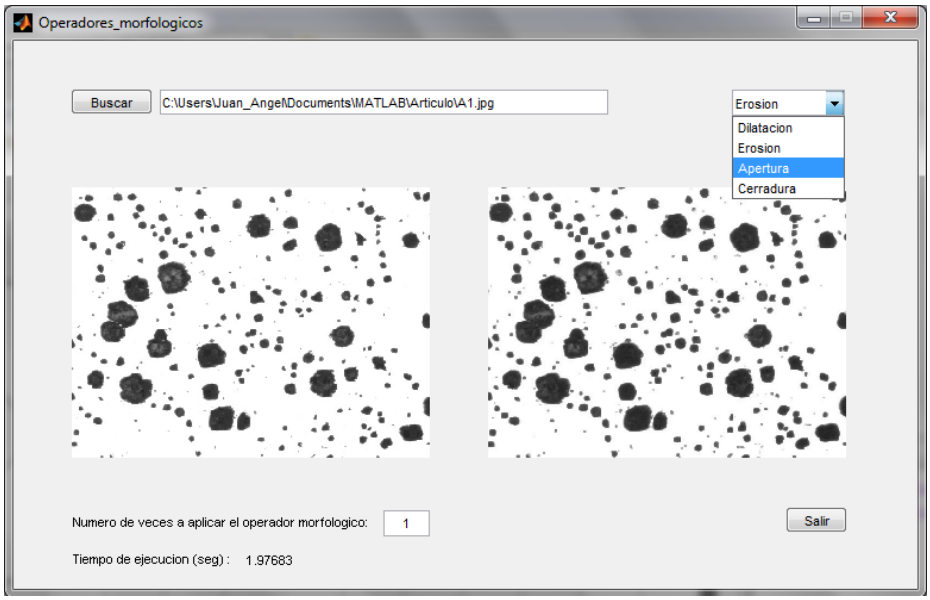
4.2 Resultados hardware-software

Debido a que él FPGA fue destinado solamente a realizar el cálculo de los valores numéricos para obtener el espectro de tamaño, dichos valores calculados son enviados a la PC con el fin de observar y analizar la gráfica de espectro de tamaños y a partir de ahí encontrar el índice de cuantificación espacial que se propone y además estimar el tamaño y la distancia máxima de los nódulos u objetos en la imagen a procesar.

4.2.1 Pruebas en software.

Para realizar las pruebas en software se utilizó el software MATLAB, se desarrolló una interfaz gráfica para el usuario (GUI, *Grapich User Interface*) con la cual es posible medir el tiempo de ejecución de los algoritmos desarrollados, en este caso, el tiempo de ejecución los operadores morfológicos, el tamaño de la imagen metalográfica utilizada es de 494 x 656 pixeles, las características de la PC donde se desarrollaron las pruebas son las siguientes: el procesador es Intel Core i5 2.30Ghz, tiene 6 GB en RAM y el sistema operativo es Windows 7 a 64 bits.

En la Figura 4-7 se muestra la GUI desarrollada para realizar las pruebas, cabe mencionar que los algoritmos de los operadores morfológicos fueron desarrollados.



4-7 GUI desarrollada.

La GUI tiene la opción de seleccionar cuál de los cuatro operadores morfológicos se desea usar, despliega el tiempo que tarda en ejecutarse los algoritmos, muestra la imagen original y la nueva imagen procesada.

Se realizaron 50 pruebas para medir el tiempo de ejecución para los diferentes operadores morfológicos, una vez obtenidos los resultados de los tiempos se realizó un promedio de los mismos, obteniéndose los siguientes resultados:

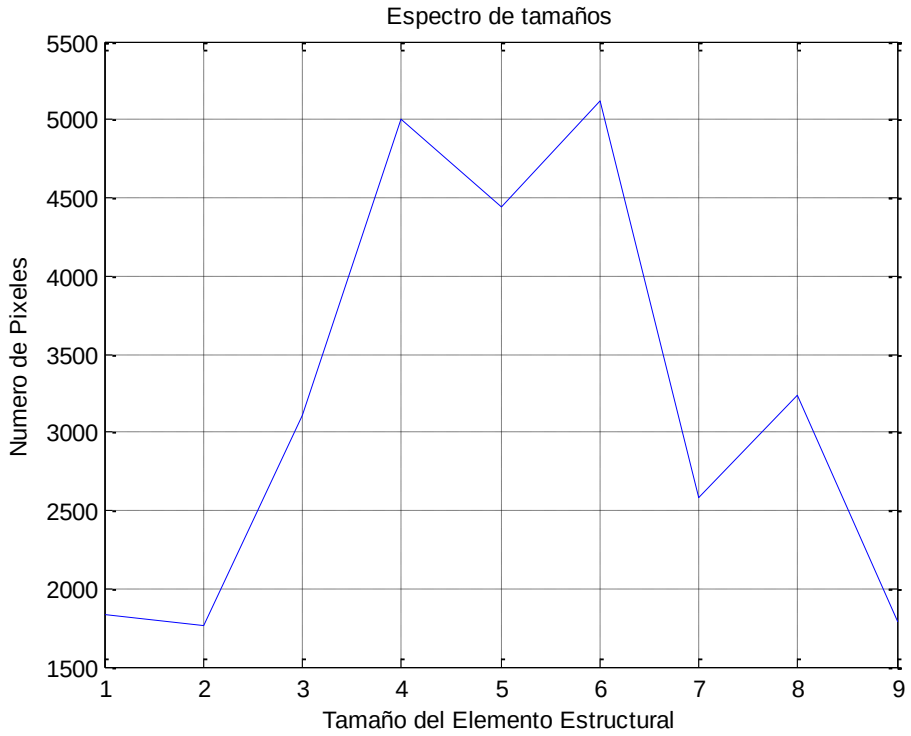
Número de veces que se aplicó el operador morfológico	1	2	5
Erosión/Dilatación	0.1718 seg	0.3436 seg	0.859 seg
Aperturas/Cerraduras	0.3410 seg	0.6671 seg	1.6358 seg

4-1 Tiempos obtenidos en software.

También, se realizaron pruebas para conocer el tiempo necesario para contar el número de pixeles, dando como resultado un tiempo de 0.006865 segundos y para hacer la copia de la imagen dando un tiempo de $9.281x10^{-7}$ segundos.

En la Figura 4-8 se muestra el cálculo del espectro de tamaño para cuando se realizan 10 aperturas, la fórmula utilizada es la siguiente:

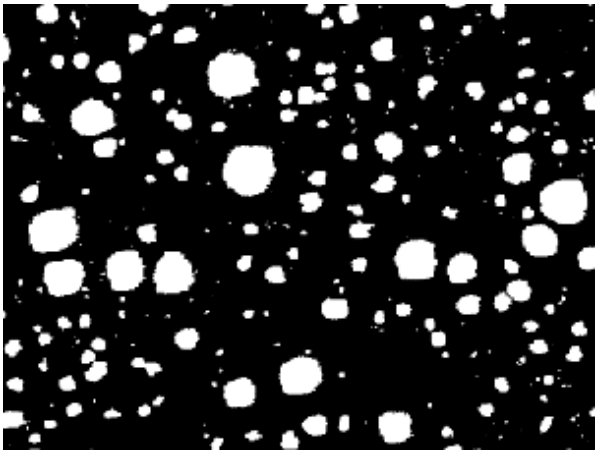
$$\mathrm{ET_{nB}(A)} = \mathrm{vol}\left(\gamma_{\mathrm{nB}}(\mathrm{A}) - \gamma_{(\mathrm{n+1})\mathrm{B}}(\mathrm{A})\right), \mathrm{n} \geq 0$$



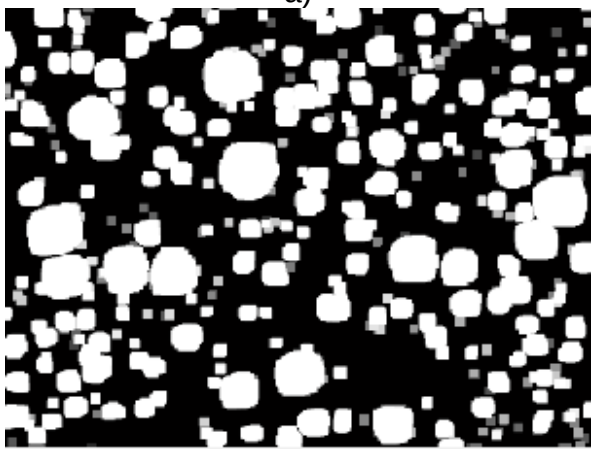
4-8 Espectro de tamaño para 10 aperturas

4.2.2 Pruebas en hardware.

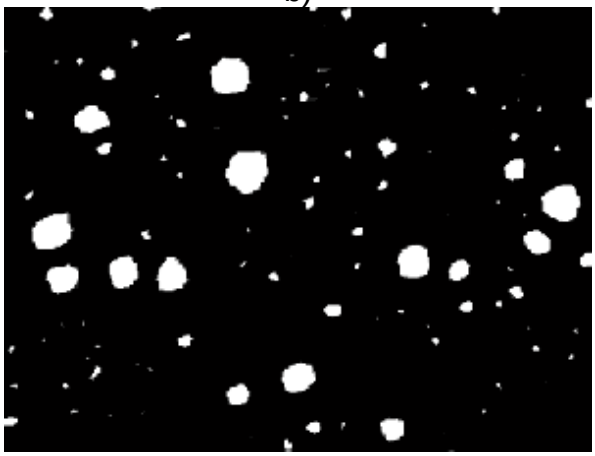
A continuación se muestran las pruebas realizadas en hardware de los operadores morfológicos de erosión y dilatación, se muestran las imágenes procesadas obtenidas en la Figura 4-9.



a)



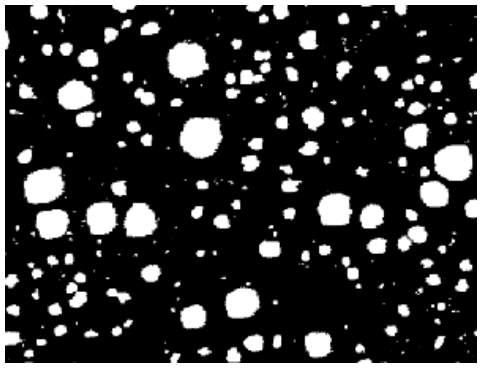
b)



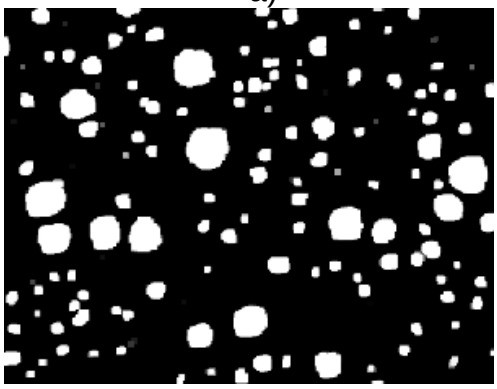
c)

4-9. a) Imagen original, b) Imagen dilatada 3, d) Imagen erosionada 3

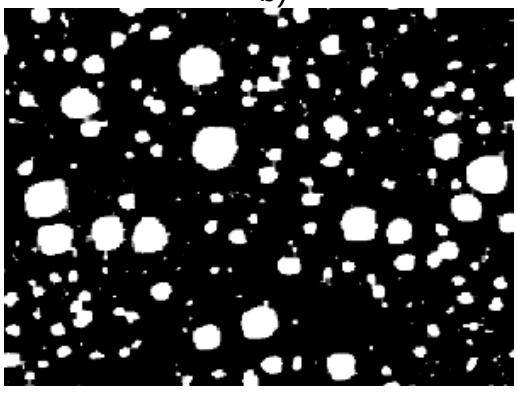
Los resultados obtenidos (Figura 4-9) para cuando se aplican los operadores morfológicos 3 veces a la imagen original fueron los mismos que los que se obtienen en software, por lo que se las descripciones digitales de los operadores morfológicos de erosión y dilatación funcionan correctamente, una vez comprobado su funcionamiento se prosiguió a realizar los operadores morfológicos de apertura y cerradura y se obtuvieron los siguientes resultados de la Figura 4-10.



a)



b)



c)

4-10.Pruebas de operadores morfológicos. a)Imagen original, b) Apertura 2, c) Cerradura 2

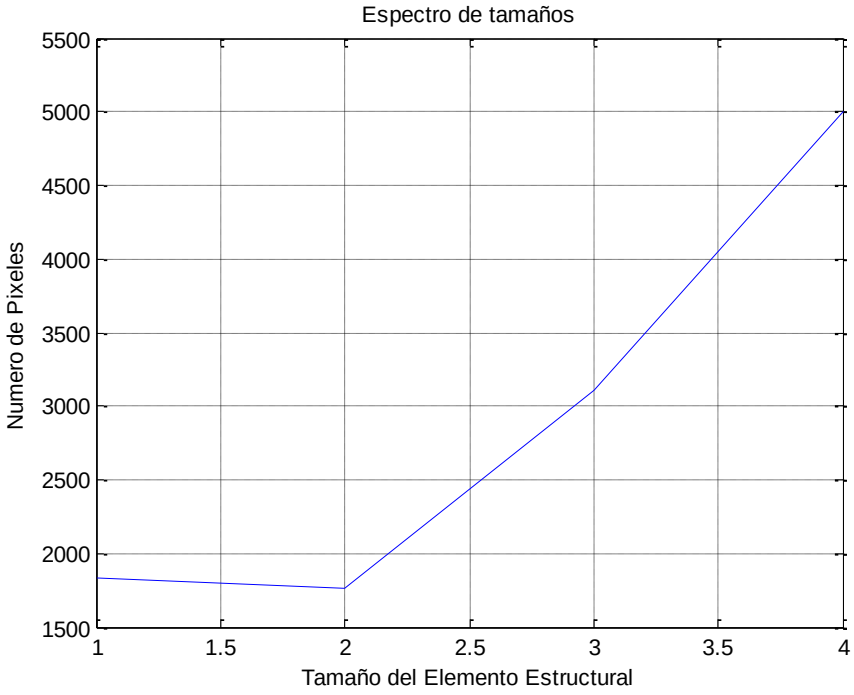
Se prosiguió a realizar las pruebas en hardware, de igual manera también se ejecutaron las pruebas 50 veces obteniendo los siguientes resultados (Tabla 4-2).

Número de veces que se aplicó el operador morfológico	**1**	**2**	**5**
Erosión/Dilatación	0.3107 seg	0.6214 seg	1.5535 seg
Aperturas/Cerraduras	0.6214 seg	1.2428 seg	3.107 seg

4-2 Tiempos obtenidos en Hardware

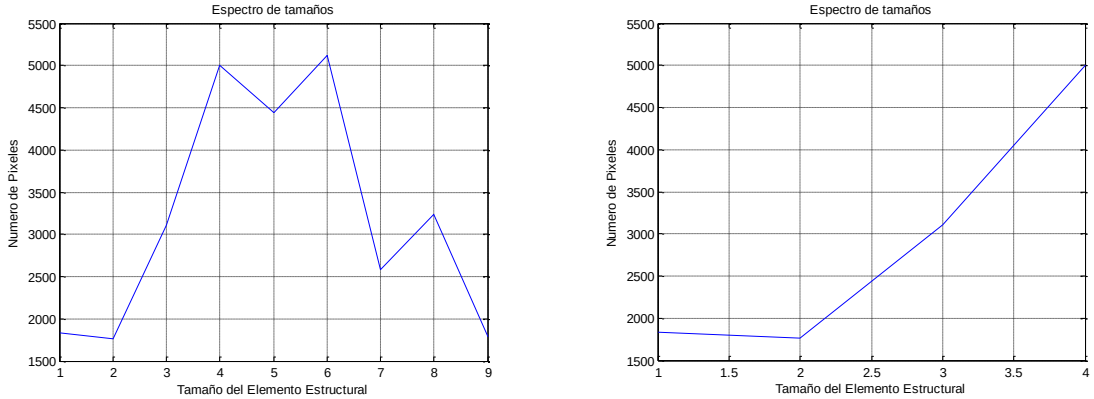
Se tomó el tiempo necesario para realizar el barrido a la imagen completa y con ello contar los pixeles de nuestro interés dando un tiempo de $7.704x10^{-6}$ segundos y un tiempo de $6.72x10^{-6}$ segundos para hacer la copia de la imagen.

Para realizar el cálculo del espectro de tamaño en hardware se realizaron 5 aperturas obteniendo los siguientes resultados de la Figura 4-11



4-11 Espectro de tamaño para 5 aperturas.

Comparando los resultados obtenidos (Figura 4-12) en hardware con respecto a los obtenidos en software se observa que efectivamente se obtiene los mismos resultandos en el cálculo de espectro de tamaño, pero el tiempo necesario en software para realizar este mismo cálculo es de 1.6358 segundos mientras que en hardware le toma 3.107 segundos.



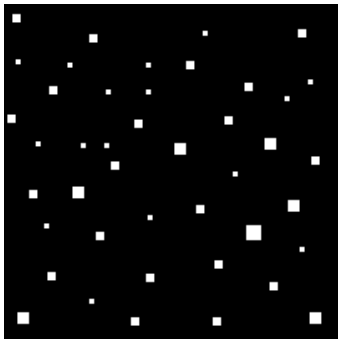
a) Resultados en Software b) Resultados en Hardware

4-12 Resultados obtenidos a) resultados en software, b) resultados en Hardware

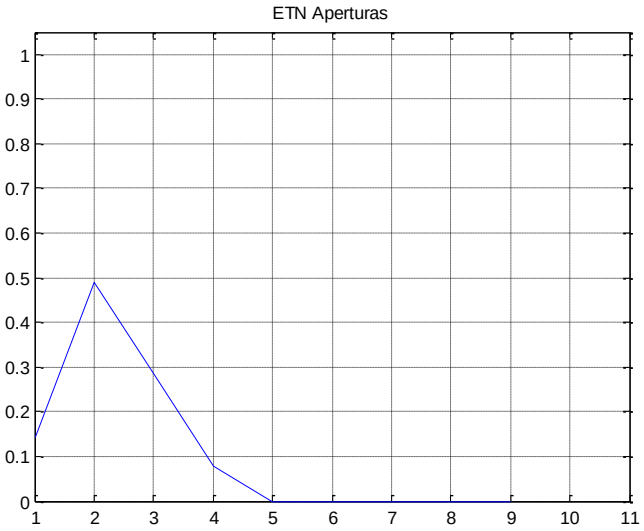
4.2.3 Pruebas en imágenes ideales.

4.2.3.1 Espectro de tamaño aplicando aperturas.

Estas pruebas fueron realizadas en software, con la intención de mostrar que es posible calcular la cantidad de nódulos en la imagen con ayuda del espectro de tamaños por aperturas, a continuación se muestran pruebas que se realizaron en imágenes sintéticas, es decir, una imagen artificial de tamaño 200x200 pixeles con nódulos de tamaño de 3, 5, 7 y 9 pixeles por lado; esta imagen fue hecha para demostrar como con ayuda del espectro de tamaño por aperturas es posible contabilizar el número de partículas o nódulos que hay en la imagen y así también poder determinar o estimar el tamaño de las mismas. En la imagen 4-13 se muestra la imagen artificial, el espectro de tamaños obtenidos y la interpretación que se da a los resultados obtenidos.



a)



b)

4-13 a) Imagen sintética, b) Espectro de tamaños normalizado utilizando Aperturas

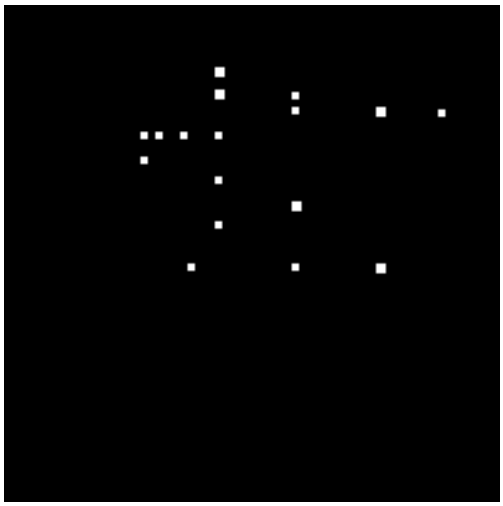
Gracias a los valores obtenidos en la gráfica de espectro de tamaños, es posible determinar la cantidad y tamaño de nódulos que hay en la imagen, por lo cual se puede obtener la cantidad de objetos que hay en la imagen, dando como resultado: 16 partículas de tamaño 3x3, 20 de 5x5, 6 de 7x7 y 1 de 9x9 pixeles. La información anterior concuerda perfectamente con la imagen sintética desarrollada, observando el b) de la Figura 4-13, al aplicar una apertura 2 se obtiene el valor de mayor magnitud en el espectro de tamaños por lo que se concluye que el tamaño de partículas que más se repiten en la imagen es de tamaño 5x5 pixeles, esto se deduce de la fórmula 4-1.

(2n+1) Ec. 4-1

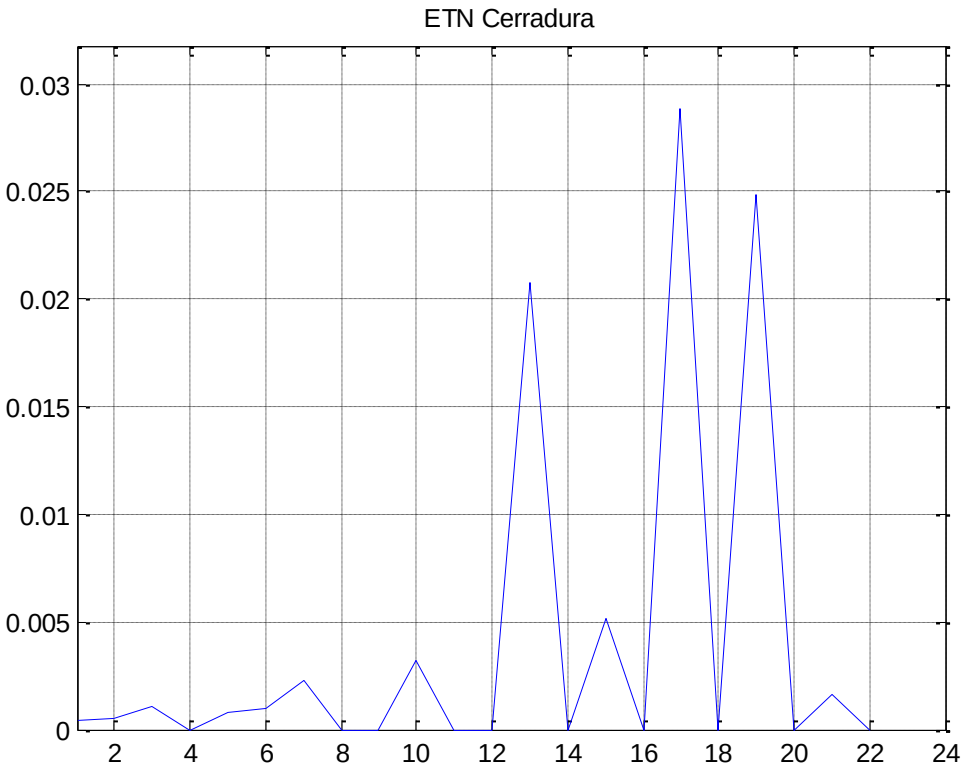
Donde *n* es el número de veces que se aplica el operador morfológico.

4.2.3.2 Espectro de tamaños aplicando cerraduras.

Con ayuda del espectro de tamaños por cerraduras es posible calcular la distancia que existe entre los nódulos de la imagen a analizar; la imagen sintética utilizada y el espectro de tamaños normalizada por cerraduras para la misma se observan en la Figura 4-14.



a)



b)

4-14 a) Imagen sintética b) Espectro de tamaño normalizado utilizando cerraduras.

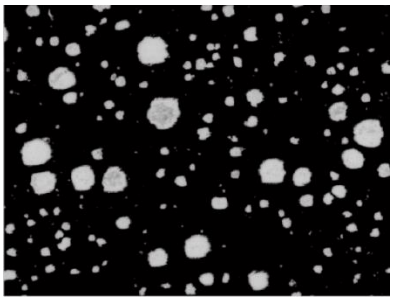
Por lo tanto, la distancia más grande encontrada en la imagen es cuando se aplica el operador morfológico 17, es decir hay una distancia máxima de 35 pixeles (ver Ec. 4-1) entre los nódulos de la imagen sintética, lo cual concuerda perfectamente con la imagen artificial.

4.3 Resultados de la integración del sensor inteligente basado en FPGA para la cuantificación de distribuciones espaciales entre objetos.

4.3.1 Cálculo de tamaño de nódulos.

Una vez que se tiene calculado el espectro de tamaño de la imagen por aperturas y cerraduras lo que prosigue es realizar el cálculo del índice de cuantificación espacial que se propone.

En la Figura 4-15 se observa la imagen de tamaño 494x656 que fue analizada.

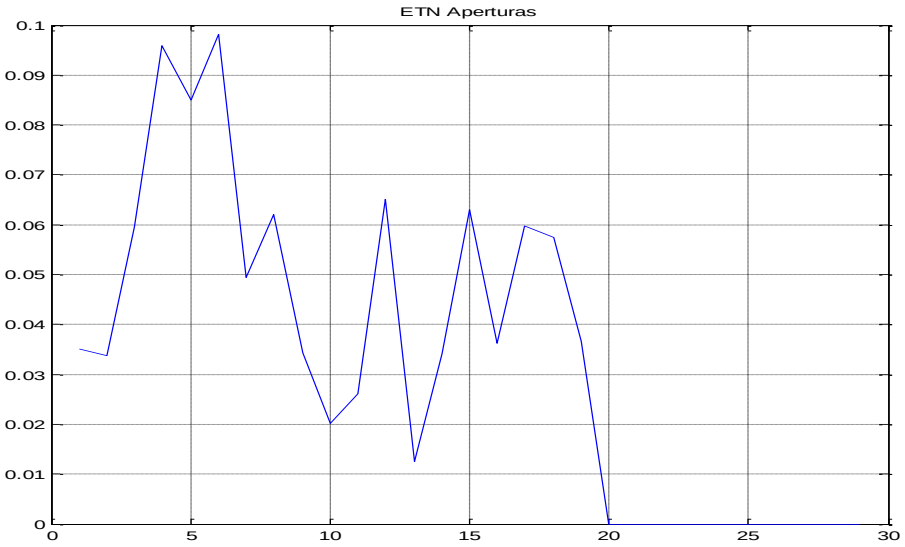


4-15 Imagen de una fundición nodular

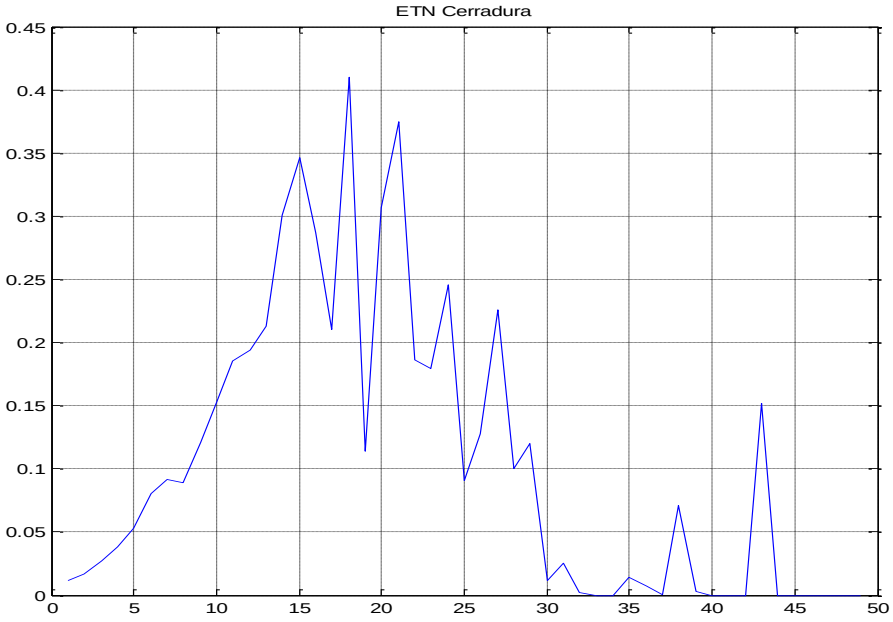
En la imagen 4-16 se muestran los espectros de tamaño normalizados por apertura y cerraduras, para calcular el índice de cuantificación que se propone es necesario seleccionar el valor de mayor intensidad obtenidos en los espectros de tamaños por apertura y cerradura, una vez seleccionados dichos valores se puede calcular el tamaño del nódulo y la distancia entre nódulos correspondientes si se desea tener mayor información de la imagen analizada.

Los resultados obtenidos de dichos análisis son 6 aperturas (nódulo del tamaño 13x13 aproximadamente) con una magnitud de 0.0982 y la distancia entre nódulos 18 cerraduras (distancia máxima de 37 pixeles) con una intensidad de 0.4102, por lo que de acuerdo a la fórmula 2.47 se obtiene el siguiente índice de cuantificación:

$$I = \frac{\gamma max}{\varphi max} = \frac{0.0982}{0.4102} = 0.2394$$



a)

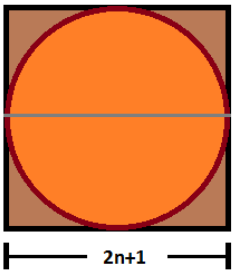


b)

4-16 a) Espectro de tamaño normalizada por aperturas, b) Espectro de tamaños normalizada por cerraduras.

La longitud de cada pixel de la imagen metalográfica es de 0.00138 mm, por lo que para el ejemplo mostrado anteriormente se hace la siguiente consideración ya que los nódulos tienen forma circular, para una máscara de n x n el área es n^2unidades cuadradas, pero debido a la forma circular del nódulo el

área utilizada por el nódulo es de πr^2, donde *r* es la mitad del valor del kernel utilizado.



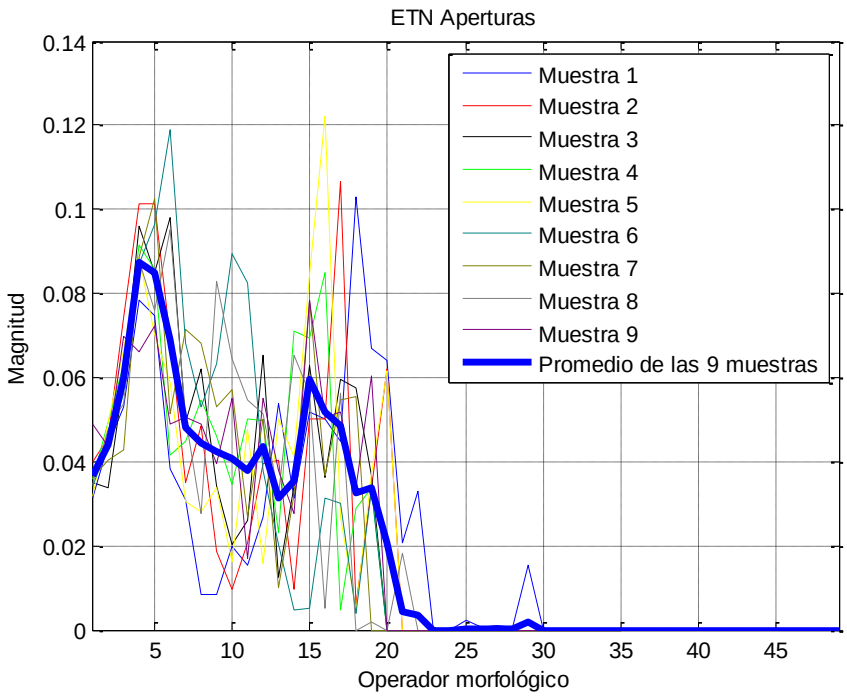
4-17 Consideración para estimar la distancia máxima entre nódulos, así como el área del mismo.

Se realizaron más pruebas a un conjunto de 10 imágenes metalográficas para poder corroborar que es posible utilizar dicho índice de cuantificación propuesto para clasificar el material analizado de acuerdo a sus propiedades físicas.

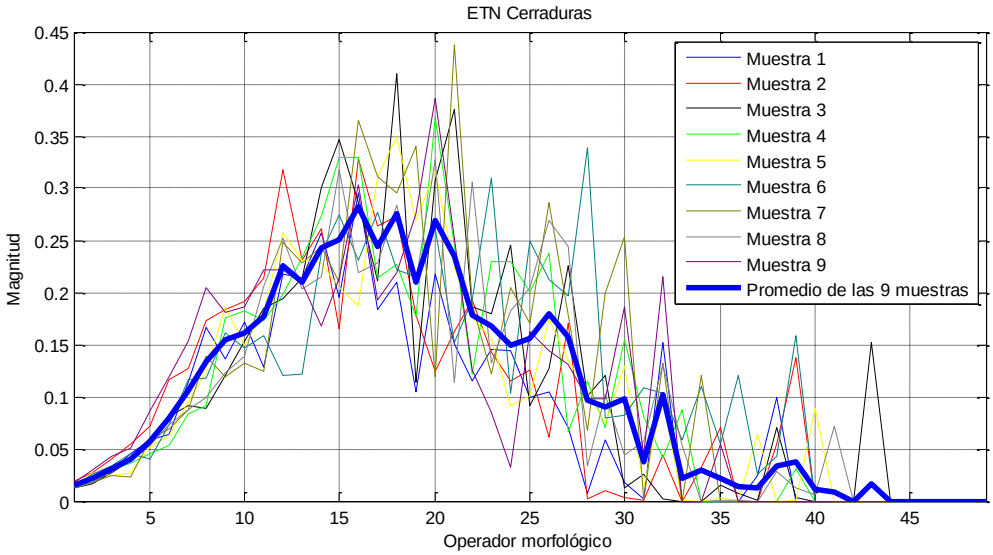
Los resultados de las intensidades de mayor magnitud en los espectros de tamaños obtenidos al analizar el conjunto de 9 imágenes de muestras de fundiciones nodulares son los siguientes:

	ETN Aperturas	ETN Cerraduras	Índice
Muestra 1	0.1029	0.2950	0.3480
Muestra 2	0.1067	0.3283	0.3251
Muestra 3	0.0982	0.4102	0.2394
Muestra 4	0.0915	0.3679	0.2488
Muestra 5	0.1223	0.3491	0.3505
Muestra 6	0.1189	0.3387	0.3510
Muestra 7	0.1024	0.4374	0.2341
Muestra 8	0.0952	0.3273	0.2908
Muestra 9	0.0786	0.3859	0.2037
Promedio			
	0.1018	0.3599	0.2879
Desviación estándar			
	0.01341	0.04488	0.05784

4-3 Datos obtenidos de los espectros de tamaños por aperturas y cerraduras para cada muestra



a)



b)

4-18 Espectros de tamaños obtenidos para el análisis de 9 muestras de fundiciones nodulares

Como se observa en las imágenes 4-18, se ve que los espectros de tamaños de las imágenes analizadas siguen un comportamiento muy similar, lo que indica que es viable utilizar el índice de cuantificación espacial que se propone en este trabajo para verificar, clasificar o en su defecto certificar que el material cumple con las características físicas con las que se fabricó.

4.4 Tiempos de procesamiento en software y hardware.

Se realizaron pruebas en varias PC's con diferentes procesadores y cantidad de memoria RAM, esto con el fin de ver como varía el tiempo de procesamiento dependiendo de las características físicas de la PC's.

Proceso realizado	AMD A6-3400M APU a 1.4 Ghz y 4 GB en RAM	AMD Fx 4300 y 8Gb RAM	i3 a 1.9 Ghz y 3.88 GB en RAM	i5 a 2.3Ghz y 6 Gb en RAM	i7 a 2.4Ghz y 16 Gb en RAM	FPGA Altera DE2-115
Copiar imagen	$9.672X10^{-7}$	$9.979X10^{-7}$	$2.28687X10^{-6}$	$9.281X10^{-7}$	$4.516X10^{-7}$	$6.72X10^{-6}$
Erosión/ Dilatación	0.2273	0.1656	0.2228	0.1814	0.1436	0.3107
Apertura/ Cerradura	0.4486	0.3345	0.4462	0.3602	0.2939	0.6215
Contar Pixeles	0.02204	0.005249	0.008572	0.006865	0.003242	$7.704X10^{-6}$
Espectro de tamaños	22 min y 50 seg	14 min y 13 seg	18 min y 37 seg	13 min y 24 seg	9 min y 32 seg	26 min y 25 seg

4-4 Comparación de tiempos en software vs hardware (Los tiempos mostrados están en segundos)

Como se observa en primera instancia el tiempo de procesamiento en el FPGA es mucho mayor que el procesamiento realizado en las PC's, pero esto se debe al diseño implementado en el FPGA, en dicho diseño solo se utilizan los 8 bits menos significativos de los 16 bits con los que cuenta cada dirección de la SRAM, por lo que al utilizar los otros 8 bits disponibles es posible calcular el espectro de tamaño por aperturas y cerraduras de manera paralela, por lo que el tiempo se reduciría a la mitad es decir, a unos 13 min con 15 segundos aproximadamente, dicho tiempo de procesamiento sería constante y no se ve afectado como sucede en las PC's donde el tiempo para calcular el espectro de tamaños para aperturas y cerraduras son afectados directamente de los procesos que se estén ejecutando en la PC.

Capítulo 5.
Conclusiones y prospectivas

Se puede concluir de acuerdo a los resultados obtenidos los siguientes puntos.

Los resultados simulados y experimentales demuestran que la implementación del cálculo de espectro de tamaños en hardware funciona de manera satisfactoria.

Los tiempos de ejecución de los operadores morfológicos implementados en hardware no se ven alterado al realizar múltiples tareas como lo que pasa en software.

Los I.P. cores de los operadores morfológicos se pueden implementar en diferentes familias de FPGA y ser acoplados a otros proyectos.

El desarrollo del sensor inteligente implementado en FPGA para realizar el cálculo de espectro de tamaño y el índice de cuantificación de las imágenes a analizar, da la opción de poder utilizar diferentes protocolos de comunicación para realizar la adquisición de los datos o imágenes.

Los sensores inteligentes basado en FPGA al realizar un proceso o cálculo de datos sin la necesidad de un PC demuestran la independencia que existe en relación al uso de un software para poder calcular el resultado.

Los sensores inteligentes basados en FPGA disminuyen considerablemente el tiempo y costo del procesamiento de datos en comparación si estos se hicieran con ayuda de un software especializado.

El costo de un proyecto disminuye considerablemente cuando se desarrollan I.P. cores.

Desarrollar el core o driver para la manipulación de datos en la DRAM (*Dynamic Random Access Memory*, Memoria Dinámica de Acceso Aleatorio) reducirá el tiempo de cálculo del índice de cuantificación espacial de una manera muy considerable.

El desarrollo de cores o drivers de protocolos de comunicación de cámaras de video permitirá tener una mayor versatilidad al momento de realizar la adquisición de imágenes.

Artículo.

System development based on FPGA to calculate the size spectrum in nodular functions used morphological operators.

Ramírez-Núñez J.A. [1], Osornio-Rios R.A. [2], Morales-Hernández L.A. [2], Morales-Velázquez L. [2], Mendoza-Galindo F.J. [1]

[1] *Maestría en Mecatrónica, Facultad de Ingeniería, Posgrado, Universidad Autónoma de Querétaro, Campus San Juan del Río, Río Moctezuma 249, San Cayetano, CP. 76807, San Juan del Río, México*

[2] *HSP Digital CA-Mecatrónica , Facultad de Ingeniería, Posgrado, Universidad Autónoma de Querétaro, Campus San Juan del Río, Río Moctezuma 249, San Cayetano, CP. 76807, San Juan del Río, México*

juan.angel.rn@hotmail.com,raor@uaq.mx,lamorales@hsdigital.org,lmorales@hspdigital.org, fmendoza@hspdigital.org

Abstract— **In nodular foundries, determining the nodular sizes (the size spectrum) of metallographic images is a work that it´s realized by software systems, although, this work uses a PC only for this task, and this routine requires a considerable computing load and a high execution time. In this work digital description of algorithm is realized, permitting to calculate the spectrum sizes and it is implemented in hardware based on an FPGA (Altera DE2-115), to minimize the computing load and decrease the execution time in a considerable way; obtaining a dedicated system to make this task efficiently and independent of a PC. The erosion and dilatation morphological operators have been used to calculate the sizes spectrum, digital description of these was made to implement the algorithm in hardware, the kernel used is a 3x3 mask, with the image sizes spectrum the maximum nodular size of the image can be determined, metallographic images testing was realized in gray scale with size of 656x494 pixels, with this, the computing load is reduced and also the processing time in comparison whit software implementations; in addition, the I.P. Cores are ours and thanks to this is possible create a smart sensor that will be used to obtain the nodule size without PC.**

Key Words --- nodular foundries, metallographic images, FPGA, sizes spectrum, I.P. Cores.

I. INTRODUCTION

Perform the image analysis, requires a massive data processing. However, this requires a big computing load and the executing times are high. Therefore, doing these processing in hardware reduces the load computing and the executing time of the same.

Reducing the processing time in software depend of the experience of the programmer ,the characteristics of the PC and the number of process that the pc will make in the moment of process the image.

In nodular foundries is necessary making the processing to metallographic image for determining the material quality of the production. These processing has high load computing and the execution time is high too, and only one PC is assigned for make this process.

Thus, developing a system implemented in FPGA with processing image cores is important. In this article will present cores development in VHDL language for obtain the size spectrum of nodular functions and the execution times are compared between software and hardware.

In these days, the companies sold software's that make the image processing, but these software's are very expensive. The alternatives are the FPGA because they are very cheap, and the information can be save and process it in the FPGA, in this case the image.

Other important aspect is that while more functions has the software more expensive is the cost, in return, development of cores for FPGA is cheap, because these are ours cores and these cores can be implemented in different families of FPGA, for example in companies like Xilinx or Altera.

The FPGA can make different task in parallel, thus, development a system based on FPGA (I.P. Cores) [1] for the image processing reduce can the

execution time, besides of this manner it will have a dedicated system for this particular task, in this case, a system for calculated the size spectrum (ET) in metallographic images without a one pc assigned for this task.

II. Size Spectrum

The companies that produce steel alloys need make analysis of the material for assure that this material is in quality standards, thus, the companies need analyze the metallographic images of the steel and determine the size spectrum of the nodules (Fig. 1).

Fig. 1 Nodule in metallographic image

For calculate the size spectrum (ET) of the nodules in the metallographic images are used the morphological operators erosion, dilatation, opening and closing, for to use the equation 1, the background of the image will be black and objects white [2],[5].

$$ET_{nB} = vol\left(\gamma_{nB}(A) - \gamma_{(n+1)B}(A)\right) \qquad (1)$$

Where:

n is the number of times to apply the operator

vol is the number of pixel white color or gray in the image

γ is a opening.

B is a kernel.

A is the image in grayscale

If the background of image is white and the objects black will be used closings.

In the Fig. 2, it show the calculation of size spectrum for ten opening.

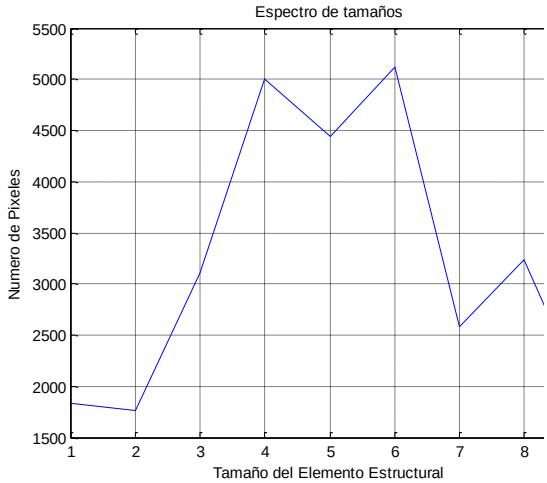


Fig. 2 Graphic of size spectrum.

III. FPGA Implementation.

The FPGA (*Field Programmable Gate Array*) are devices in where is possible implement digital descriptions of algorithms or tasks, these tasks are executed in parallel and fast. Thus, develop cores for a system of image processing in FPGA will allow decrease the execution time and the computing load to PC.

The FPGA used is the model DE2-115 (Fig.4) of the Altera family, this card was used in particular because it has big capacity SRAM *Static Random Access Memory*) [6].

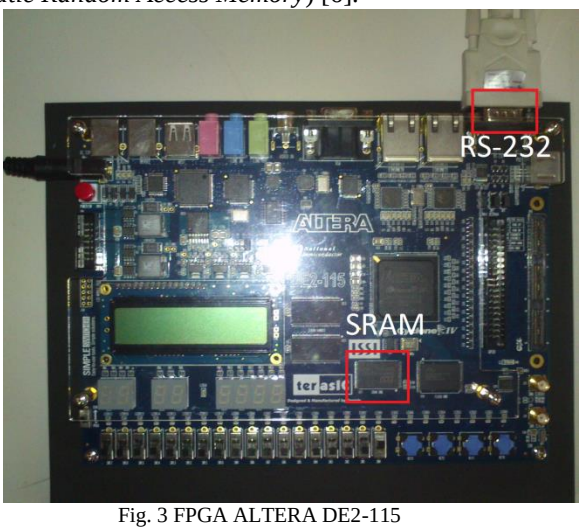
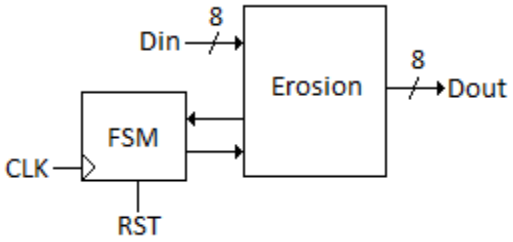


Fig. 3 FPGA ALTERA DE2-115

The digital descriptions were done in the software Active-HDL, the modules of erosion, dilation and one module for count pixels were developed (Fig.4).

The modules of serial communication RS-232 [3] were utilized for the reception of the image in the FPGA and send the results of the calculation of size spectrum to PC.



a)

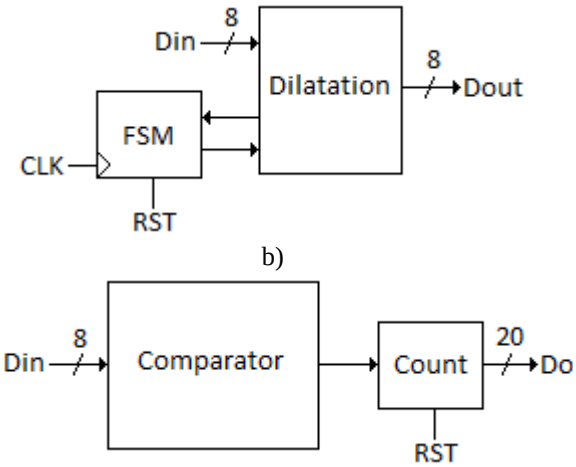


Fig.4 Modules developed: a) Erosion, b) Dilatation, c) Pixels count.

In the Fig. 5, it show the methodology implemented to do the calculation of size spectrum of the metallographic image in FPGA.

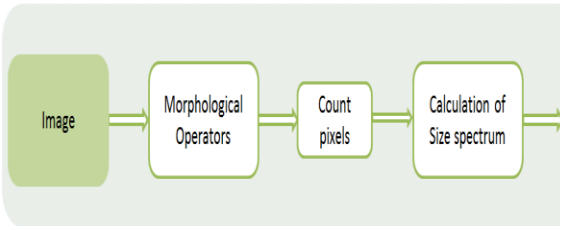


Fig.5 Methodology.

IV. Results

A. *Calculation Software.*

The image utilized has a size of 656x494 pixels (Fig. 6), the GUI (*Graphical User Interface*) was developed in the software MATLAB (*MATtrix LABoratory*) (Fig. 7), the codes of the morphological operators were developed too, the execution times of the morphological operators were measured in software with different computing loads and the kernel used is of 3 x 3 with values of ones.

The characteristics of PC are the processor is Intel Core i5 to 2.3 GHz, 6GB in RAM and a operating system of 64 bits.

In the Table I, it shows the execution time for the different morphological operators.

TABLE I

Number of times that the operator morphological was applied.	1	2	5
Erosion/Dilatatio n	1.9801se g	3.9703se g	10.1719se g
Opening/Closing	4.0121se g	7.9819se g	19.7790se g

The time in software, for count the number of pixels of the image is 0.00998 seconds.

The best execution time in PC for do one erosion or dilatation is 1.9801 seconds, of this manner for one opening or closing, the expected run time is the double (3.9602 seconds), but the PC does more task and exist an increase in the execution time.

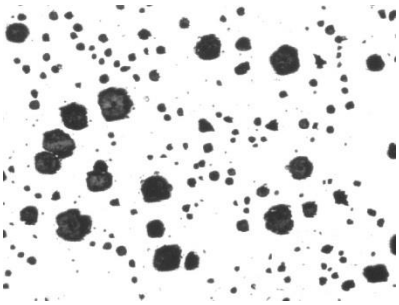


Fig. 6 Metallographic image (656 x 494).

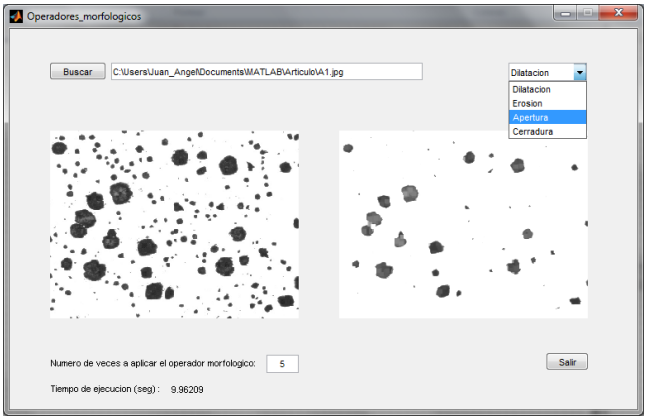


Fig. 7 Interface developed in MATLAB to measure the execution times of the morphological operators.

B. *Calculation Hardware*

The times obtained in hardware for the operators morphological are shown in the Table II.

TABLE II

Number of times that the operator morphological was applied.	1	2	5
Erosion/Dilatation	1.16seg	2.32seg	5.8seg
Opening/Closing	2.32seg	4.64seg	11.6seg

The time in hardware, for count the number of pixels of the image is 0.05185 seconds.

The Fig. 8, tt show the results of the sizes spectrum obtained in hardware. If the results are compared between software and hardware are the same results.

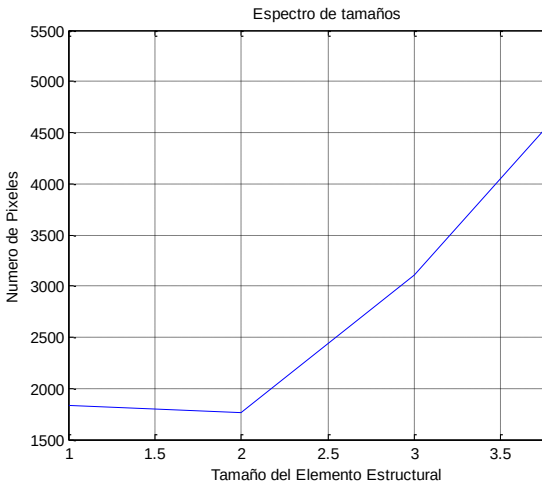


Fig.8 Results obtained in hardware.

V. Conclusions

The results simulated and experimental of the calculation of size spectrum are sames, it shows that the proposed implementation operates correctly.

Thanks to the results obtained we can conclude the following.

- The execution time of the morphological operators in hardware is more efficient than the implemented in software.
- The I.P. Cores, of the morphological operators can be implemented in different FPGA families and these modules can be used in others projects, too.
- It can develop a smart sensor implemented in FPGA for perform the calculation of size spectrum of the metallographic images without use to one PC.

References

[1] González, M. 2010. Procesador Morfológico de Imágenes FPGA aplicado al análisis metalográfico. Universidad Autónoma de Querétaro. Facultad de Ingeniería. Tesis de maestría.

[2] Morales, L. 2009. Granulometrías y segmentación de imágenes: aplicación a la caracterización de micro-estructuras en materiales y huellas digitales. Universidad Autónoma de Querétaro. Tesis Doctoral.

[3] Romero, R. 2007. Electrónica Digital y Lógica programable. Universidad de Guanajuato.

[4] 4. Ramos, A. 2009. Metodología de una etapa básica de un sistema de procesamiento de imágenes basado en FPGA, Facultad de Informática Universidad Autónoma de Querétaro.

[5] 5. L.A. Morales-Hernández, I.R. Terol-Villalobos, A.Domínguez-González, F. Manríquez-Guerrero, G. Herrera-Ruiz (2010). Spatial distribution and spheroidicity characterization of graphite nodules based on morphological tools. Journal of Materials Processing Technology 2010 pp. 335-342.

[6] C.A. Ramos Arreguín, "Metodología para Almacenamiento de Imágenes en Memorias externas de tipo RAM, emplenado FPGA". IX Congreso Internacional Sobre Innovación y Desarrollo Tecnológico (CIINDET); Cuernavaca, Morelos, México;2011.

[7] Behrooz Parhami, "Computer Arithmetic Algorithms and Hardware designs", pp. 361-373.

[8] Israle Koren, "Computer Arithmetics Algorithms", 2nd Edition, A. K. Peters, Ltd. Pp. 225-245

[9] J.Michel Muller, "Elementary Functions Algorithms and Implementation" pp. 101-123

[10] J. C. Russ, "The Image Processing Handbook", Raleigh, NC, USA. CRC PRss,2011.
R. Jain,R. Kasturi,B.G Schunck,"Machine Vision", McGrawll-Hill Series in Computer Science.

Bibliografía

Contreras, L. 2012. Smart Sensor for symptoms quantification in diseased plants. Morphological Image Processing, Megha Goyal. IJCST Vol. 2, Iss ue 4, Oct. - Dec. 2011

Cordova, D. 2010. Plataforma para la implementación de algoritmos de Procesamiento digital de imágenes en arquitecturas reconfigurables, Depto. de Ingeniería Eléctrica. Zacatecas, Zacatecas.

Gervais, S. 2011. Next Smart Sensors Generation, Sensors Applications Symposium (SAS)

González, M. 2010. Procesador Morfológico de Imágenes FPGA aplicado al análisis metalográfico. Universidad Autónoma de Querétaro. Facultad de Ingeniería. Tesis de maestría.

Morales, L. 2009. Granulometrías y segmentación de imágenes: aplicación a la caracterización de micro-estructuras en materiales y huellas digitales. Universidad Autónoma de Querétaro. Tesis Doctoral.

Morales, L. 2009. New Directional Morphological Approaches for the characterization of fingerprints. Ingeniería, Investigación y Tecnología de la UNAM, 10,257-259.

Morales, L. 2009. Graphite nodules characterization using mathematical morphology techniques. Acta Microscópica, 18, 174-184.

Morales, L. 2010. Spatial distribution and spheroidicity characterization of graphite nodules based on morphological tools. Journal of Materials Processing technology.

Morphological Edge Detection Algorithm Based on Multi-Structure Elements of Different Directions .1 C.NagaRaju ,2 S.NagaMani, 3 G.rakesh Prasad, 4 S.Sunitha. Volume 1 No. 1, May 2011 International Journal of Information and Communication Technology Research.

Morphological Image Processing: Basic Concepts. Spring 2009 ELEN 4304/5365. Gleb V. Tcheslavski.

Press Release. Presentation Speech. Godfrey N Hounsfield-Autobiography. Nobel Prize in Physiology or Medicine. October 1979

Romero, R. 2007. Electrónica Digital y Lógica programable. Universidad de Guanajuato.

Ramos, A. 2009. Metodología de una etapa básica de un sistema de procesamiento de imágenes basado en FPGA, Facultad de Informática Universidad Autónoma de Querétaro.

Rodriguez, C. 2011. Fused Smart senso network for multi- axis forward kinematics estimation in industrial robots.

Texture classification by statistical learning from morphological image processing: application to metallic surfaces. Journal ofMicroscopy, Vol. 239, Pt 2 2010, pp. 159–166.

Trejo, M. 2010. FPGA-Based Smart Sensor for Online Displacement Measurements Using a Heterodyne Interferometer.

Using Mathematical Morphology to Detect the Imperfections of the Printed Circuit Boards. I. Balan. Journal of Applied Computer Science, no.1 (2) /2008, Suceava.

Morphological Edge Detection Algorithm Based on Multi-Structure Elements of Different Directions .1 C.NagaRaju ,2 S.NagaMani, 3 G.rakesh Prasad, 4 S.Sunitha. Volume 1 No. 1, May 2011 International Journal of Information and Communication Technology Research.

Morphological Image Processing: Basic Concepts. Spring 2009 ELEN 4304/5365. Gleb V. Tcheslavski.

Romero, R. 2007. Electrónica Digital y Lógica programable. Universidad de Guanajuato.

Ramos, A. 2009. Metodología de una etapa básica de un sistema de procesamiento de imágenes basado en FPGA, Facultad de Informática Universidad Autónoma de Querétaro.

Rodriguez, C. 2011. Fused Smart senso network for multi- axis forward kinematics estimation in industrial robots.

Texture classification by statistical learning from morphological image processing: application to metallic surfaces. Journal ofMicroscopy, Vol. 239, Pt 2 2010, pp. 159–166.

Trejo, M. 2010. FPGA-Based Smart Sensor for Online Displacement Measurements Using a Heterodyne Interferometer.

Using Mathematical Morphology to Detect the Imperfections of the Printed Circuit Boards. I. Balan. Journal of Applied Computer Science, no.1 (2) /2008, Suceava.

Apendices.

SRAM.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SRAM is
   Port (
                              RST   : in STD_LOGIC;
                              CLK   : in STD_LOGIC;
                              Di    : in STD_LOGIC_VECTOR (15 downto 0);
                              WR    : in STD_LOGIC;
                              ADDR  : in STD_LOGIC_VECTOR (19 downto 0);
                               D          : inout STD_LOGIC_VECTOR (15 downto 0);
                              A     : out STD_LOGIC_VECTOR (19 downto 0);
                              CS    : out STD_LOGIC;
                              OE    : out STD_LOGIC;
                              WE    : out STD_LOGIC;
                              UB    : out STD_LOGIC;
                              LB    : out STD_LOGIC;
      Do    : out STD_LOGIC_VECTOR (15 downto 0)
                              );
end SRAM;

architecture Behavioral of SRAM is

begin

          D <= Di when (WR = '1') else (others => 'Z');
          Do <= (others => '0') when (RST = '0') else D when rising_edge(CLK);
          WE <= NOT (WR);
          OE <= WR;
          CS <= '0';
          UB <= '0';
          LB <= '0';
          A <= ADDR;

end Behavioral;
```

Copiar imagen.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity COPIAR_IMAGEN is
port(
        RST     : in std_logic;
        CLK     : in std_logic;
        STT     : in std_logic;
        R               : in std_logic_vector(19 downto 0);
        C               : in std_logic_vector(19 downto 0);
        DIR     : in std_logic_vector(19 downto 0);
        DATI    : in std_logic_vector(15 downto 0);
        WE              : out std_logic;
        DAT     : out std_logic_vector(15 downto 0);
        ADD     : out std_logic_vector(19 downto 0);
        FIN     : out std_logic
);
end COPIAR_IMAGEN;

architecture proceso_ci of COPIAR_IMAGEN is

component FSM_COPIAR2
port(
        RST     : in std_logic;
        CLK     : in std_logic;
        STT     : in std_logic;
        FDEC  : in std_logic;
        WE              : out std_logic;
        R1              : out std_logic;
        FADD    : out std_logic;
        OPC     : out std_logic_vector(1 downto 0);
        FIN     : out std_logic
);
end component;

component contador_multifuncional
        generic(
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q         : out std_logic_vector(n-1 downto 0)
        );
end component;

signal ADD1,ADD2,RECO: std_logic_vector(19 downto 0);
```

```
signal DAT1                                 : std_logic_vector(15 downto 0);
signal R1,FADD,FDEC      : std_logic;
signal OPC                                  : std_logic_vector(1 downto 0);

begin
RECO <= std_logic_vector(unsigned(r(9 downto 0))*unsigned(c(9 downto 0)));
bloque_00: FSM_COPIAR2 port map(RST,CLK,STT,FDEC,WE,R1,FADD,OPC,FIN);       -- WE
bloque_01: contador_multifuncional generic map(20) port map(RST,CLK,OPC,ADD1);
ADD2 <= std_logic_vector(unsigned(ADD1)+unsigned(DIR));
ADD  <= ADD1 when FADD='0' else ADD2;            -- DIRECCIONES
DAT1 <= (others=>'0') when (RST='0') else DATI when rising_edge(CLK) AND R1='1';
DAT  <= "00000000"&(DAT1(7 downto 0));           -- DATOS
FDEC <= '0' when ADD1(9 downto 0) < RECO(9 downto 0) else '1';

end proceso_ci;
```

DEMUX 4-1.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity DEMUX_41 is
        generic(
        n: integer:= 16
        );
        port(
        A       : in std_logic_vector(n-1 downto 0);
        SEL: in std_logic_vector(1 downto 0);
        B1      : out std_logic_vector(n-1 downto 0);
        B2      : out std_logic_vector(n-1 downto 0);
        B3      : out std_logic_vector(n-1 downto 0);
        B4      : out std_logic_vector(n-1 downto 0)
        );
end DEMUX_41;

architecture decision of DEMUX_41 is
begin
        uno: process(A,SEL)
        begin
        case SEL is
                when "00" =>
                B1 <= A;
                B2 <= (others => '0');
                B3 <= (others => '0');
                B4 <= (others => '0');
                when "01" =>
                B1 <= (others => '0');
                B2 <= A;
                B3 <= (others => '0');
                B4 <= (others => '0');
```

```
                when "10" =>
                B1 <= (others => '0');
                B2 <= (others => '0');
                B3 <= A;
                B4 <= (others => '0');
                when others =>
                B1 <= (others => '0');
                B2 <= (others => '0');
                B3 <= (others => '0');
                B4 <= A;
        end case;
        end process uno;
end decision;
```

Contador multifuncional

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity contador_multifuncional is
        generic(
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end contador_multifuncional;

architecture Estructural of contador_multifuncional is

signal Qn,Qp: std_logic_vector(n-1 downto 0);

begin

        combinacional: process(Qp,OPC)
        begin
                case OPC is
                        when "00"   => Qn <= (others=>'0');             -- Borrar
                        when "01"   => Qn <= Qp;                                -- Mantener
                        when "10"   => Qn <= Qp+1;                              -- Incrementar
                        when others => Qn <= Qp-1;                              -- Decrementar
                end case;
                Q <= Qp;
        end process combinacional;
```

```
--	secuencial: process(RST,CLK)
--	begin
--		if(RST='0')then
--			Qp <= (others=>'0');
--		elsif(CLK'event and CLK='1')then
--			Qp <= Qn;
--		end if;
--	end process secuencial;
	Qp <= (others => '0') when (RST='0') else Qn when rising_edge(CLK);
end Estructural;
```

Izquierda Derecha

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity izquierda_derecha_nueva is
	port(
	RST	: in std_logic;                                          -- Senal de reset
	CLK	: in std_logic;                                          -- Senal de reloj
	STT	: in std_logic;                                          -- Senal de inicio de
procesamiento
	DIR	: in std_logic_vector(19 downto 0);              -- Direccion de inicio de
manipualcion de imagen(Renglon*Columna)
	R	: in std_logic_vector(19 downto 0);              -- Numero de Renglones
	C	: in std_logic_vector(19 downto 0);              -- Numero de Columnas
	DATI: in std_logic_vector(15 downto 0);          -- Dato de entrada
	DATO: out std_logic_vector(15 downto 0);         -- Dato de salida
	WE	: out std_logic;                                         -- Senal de lectura('0') y
escritura('1')
	ADD	: out std_logic_vector(19 downto 0);     -- Direccion donde se desea leer o escribir
	FIN	: out std_logic                                          -- Senal de fin de
procesamiento
	);
end izquierda_derecha_nueva;

architecture procesos of izquierda_derecha_nueva is

component FSM_izquierda_derecha_nueva
	port(
	RST	: in std_logic;                                  -- Senal de reset
	CLK	: in std_logic;                                  -- Senal de reloj
	STT	: in std_logic;                                  -- Senal de inicio de proceso
	FC1 : in std_logic;                                  -- Senal de bandera
columnas 1
	FR1 : in std_logic;                                  -- Senal de bandera
renglones 1
	COMP: in std_logic_vector(1 downto 0);  -- Senal de comparaciones(LE&GE);
	WE	: out std_logic;                         -- Senal de lectura('0') y escritura('1')
```

```
        R1      : out std_logic;                                    -- Senal registro 1
        R2      : out std_logic;                                    -- Senal registro 2
        OPC1: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Renglones)
        OPC2: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Columanas)
        FADD: out std_logic;                                        -- Bandera de direccion proceso uno
        FCOM: out std_logic_vector(1 downto 0);-- Bandera indicadora de proceso actual
        FIN     : out std_logic                                             -- Senal de fin de proceso
        );
end component;

component contador_multifuncional
        generic(
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

component registro
        generic(
        n: integer:=16
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        H       : in std_logic;
        D       : in std_logic_vector(n-1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

-- Senales genericas
signal OPC1,OPC2: std_logic_vector(1 downto 0);
signal CONTADOR1,CONTADOR2: std_logic_vector(19 downto 0);
signal DATMAYOR: std_logic_vector(7 downto 0);
signal G1: std_logic;
signal DAT1,DAT2: std_logic_vector(7 downto 0);
signal R1,R2: std_logic;
signal FADD: std_logic;
signal ADD1_01,ADD2_01,ADD3_01,ADD4_01: std_logic_vector(19 downto 0);
signal FC1,FR1: std_logic;
signal RESTACOLUMNA: std_logic_vector(19 downto 0);
signal COMP,FCOM: std_logic_vector(1 downto 0);
begin
```

```
        --
************************************************************************************
************************
        -- FSM


        --
************************************************************************************
************************
        bloque_33: FSM_izquierda_derecha_nueva port
map(RST,CLK,STT,FC1,FR1,COMP,WE,R1,R2,OPC1,OPC2,FADD,FCOM,FIN);
        --
************************************************************************************
************************
        -- Contadores
        --
************************************************************************************
************************
        bloque_00: contador_multifuncional generic map(20) port map(RST,CLK,OPC1,CONTADOR1);
        bloque_01: contador_multifuncional generic map(20) port map(RST,CLK,OPC2,CONTADOR2);
        --
************************************************************************************
************************
        --          Izquierda_Derecha
        --
************************************************************************************
************************
        RESTACOLUMNA <= std_logic_vector(unsigned(C)-1);
        FR1 <= '0' when CONTADOR1(9 downto 0) < R(9 downto 0) else '1';
        FC1 <= '0' when CONTADOR2(9 downto 0) < RESTACOLUMNA(9 downto 0) else '1';
        ADD4_01 <= std_logic_vector(unsigned(CONTADOR1(9 downto 0))*unsigned(C(9 downto 0)));
        ADD3_01 <= std_logic_vector(unsigned(CONTADOR2)+unsigned(ADD4_01));
        ADD1_01 <= std_logic_vector(unsigned(ADD3_01)+unsigned(DIR));
        ADD2_01 <= std_logic_vector(unsigned(ADD1_01)+1);
        ADD     <= ADD1_01 when (FADD='0') else ADD2_01;
        --
************************************************************************************
        -- Datos de salida
--************************************************************************************
        bloque_04: registro generic map(8) port map(RST,CLK,R1,DATI(7 downto 0),DAT1);
        bloque_05: registro generic map(8) port map(RST,CLK,R2,DATI(7 downto 0),DAT2);
        G1 <= '1' when DAT1 > DAT2 else '0';
        COMP <= '0'&G1;
        DATMAYOR <= DAT1 when (FCOM(0)='1') else DAT2;
        DATO <= "00000000"&DATMAYOR;
end procesos;


library IEEE;
```

Arriba Abajo.

```
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity arriba_abajo_nueva is
	port(
	RST	: in std_logic;                                          -- Senal de reset
	CLK	: in std_logic;                                          -- Senal de reloj
	STT	: in std_logic;                                          -- Senal de inicio de
procesamiento
	DIR	: in std_logic_vector(19 downto 0);              -- Direccion de inicio de
manipualcion de imagen(Renglon*Columna)
	R	: in std_logic_vector(19 downto 0);              -- Numero de Renglones
	C	: in std_logic_vector(19 downto 0);              -- Numero de Columnas
	DATI: in std_logic_vector(15 downto 0);              -- Dato de entrada
	DATO: out std_logic_vector(15 downto 0);       -- Dato de salida
	WE	: out std_logic;                                         -- Senal de lectura('0') y
escritura('1')
	ADD	: out std_logic_vector(19 downto 0);    -- Direccion donde se desea leer o escribir
	FIN	: out std_logic                                          -- Senal de fin de
procesamiento
	);
end arriba_abajo_nueva;

architecture procesos of arriba_abajo_nueva is

component FSM_arriba_abajo_nueva
	port(
	RST	: in std_logic;                                  -- Senal de reset
	CLK	: in std_logic;                                  -- Senal de reloj
	STT	: in std_logic;                                  -- Senal de inicio de
proceso
	FC1 : in std_logic;                                  -- Senal de bandera
columnas 1
	FR1 : in std_logic;                                  -- Senal de bandera
renglones 1
	COMP: in std_logic_vector(1 downto 0);         -- Senal de comparaciones(LE&GE);
	WE	: out std_logic;                                 -- Senal de lectura('0') y
escritura('1')
	R1	: out std_logic;                                 -- Senal registro 1
	R2	: out std_logic;                                 -- Senal registro 2
	OPC1: out std_logic_vector(1 downto 0);         -- Opcion contador
multifuncional(Renglones)
	OPC2: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Columanas)
	FADD: out std_logic;                                 -- Bandera de direccion proceso
uno
	FCOM: out std_logic_vector(1 downto 0);         -- Bandera indicadora de proceso actual
	FIN	: out std_logic                                  -- Senal de fin de proceso
```

```
	);
end component;

component contador_multifuncional
	generic(
	n: integer:= 20
	);
	port(
	RST	: in std_logic;
	CLK	: in std_logic;
	OPC	: in std_logic_vector(1 downto 0);
	Q	: out std_logic_vector(n-1 downto 0)
	);
end component;

component registro
	generic(
	n: integer:=16
	);
	port(
	RST	: in std_logic;
	CLK	: in std_logic;
	H	: in std_logic;
	D	: in std_logic_vector(n-1 downto 0);
	Q	: out std_logic_vector(n-1 downto 0)
	);
end component;

-- Senales genericas
signal OPC1,OPC2: std_logic_vector(1 downto 0);
signal CONTADOR1,CONTADOR2: std_logic_vector(19 downto 0);
signal DATMAYOR: std_logic_vector(7 downto 0);
signal G1: std_logic;
signal DAT1,DAT2: std_logic_vector(7 downto 0);
signal R1,R2: std_logic;
signal FADD: std_logic;
signal ADD1_01,ADD2_01,ADD3_01,ADD4_01: std_logic_vector(19 downto 0);
signal FC1,FR1: std_logic;
signal RESTARENGLON: std_logic_vector(19 downto 0);
signal COMP,FCOM: std_logic_vector(1 downto 0);
begin
	--
**************************************************************************
****************************
	-- FSM
```

```
        --
********************************************************************************
*****************************
        bloque_33: FSM_arriba_abajo_nueva port
map(RST,CLK,STT,FC1,FR1,COMP,WE,R1,R2,OPC1,OPC2,FADD,FCOM,FIN);
        --
********************************************************************************
*****************************
        -- Contadores
        --
********************************************************************************
*****************************
        bloque_00: contador_multifuncional generic map(20) port
map(RST,CLK,OPC1,CONTADOR1);
        bloque_01: contador_multifuncional generic map(20) port
map(RST,CLK,OPC2,CONTADOR2);
        --
********************************************************************************
*****************************
        --          Izquierda_Derecha
        --
********************************************************************************
*****************************
        RESTARENGLON <= std_logic_vector(unsigned(R)-1);
        FR1 <= '0' when CONTADOR1(9 downto 0) < RESTARENGLON(9 downto 0) else '1';
        FC1 <= '0' when CONTADOR2(9 downto 0) < C(9 downto 0) else '1';
        ADD4_01 <= std_logic_vector(unsigned(CONTADOR1(9 downto 0))*unsigned(C(9 downto
0)));
        ADD3_01 <= std_logic_vector(unsigned(CONTADOR2)+unsigned(ADD4_01));
        ADD1_01 <= std_logic_vector(unsigned(ADD3_01)+unsigned(DIR));
        ADD2_01 <= std_logic_vector(unsigned(ADD1_01)+unsigned(C));
        ADD     <= ADD1_01 when FADD='0' else ADD2_01;
        --
********************************************************************************
*****************************
        -- Datos de salida
        --
********************************************************************************
*****************************
        bloque_04: registro generic map(8) port map(RST,CLK,R1,DATI(7 downto 0),DAT1);
        bloque_05: registro generic map(8) port map(RST,CLK,R2,DATI(7 downto 0),DAT2);
        G1 <= '1' when DAT1 > DAT2 else '0';
        COMP <= '0'&G1;
        DATMAYOR <= DAT1 when (FCOM(0)='1') else DAT2;
        DATO <= "00000000"&DATMAYOR;
end procesos;
```

Derecha Izquierda.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity derecha_izquierda_nueva is
	port(
	RST	: in std_logic;                                                    -- Senal de reset
	CLK	: in std_logic;                                                    -- Senal de reloj
	STT	: in std_logic;                                                    -- Senal de inicio de
procesamiento
	DIR	: in std_logic_vector(19 downto 0);          -- Direccion de inicio de
manipualcion de imagen(Renglon*Columna)
	R	: in std_logic_vector(19 downto 0);          -- Numero de Renglones
	C	: in std_logic_vector(19 downto 0);          -- Numero de Columnas
	DATI: in std_logic_vector(15 downto 0);      -- Dato de entrada
	DATO: out std_logic_vector(15 downto 0);     -- Dato de salida
	WE	: out std_logic;                                           -- Senal de lectura('0') y
escritura('1')
	ADD	: out std_logic_vector(19 downto 0);     -- Direccion donde se desea leer o escribir
	FIN	: out std_logic                                                    -- Senal de fin de
procesamiento
	);
end derecha_izquierda_nueva;

architecture procesos of derecha_izquierda_nueva is

component FSM_derecha_izquierda_nueva
	port(
	RST	: in std_logic;                                            -- Senal de reset
	CLK	: in std_logic;                                            -- Senal de reloj
	STT	: in std_logic;                                            -- Senal de inicio de proceso
	FC1 : in std_logic;                                            -- Senal de bandera
columnas 1
	FR1 : in std_logic;                                            -- Senal de bandera
renglones 1
	COMP: in std_logic_vector(1 downto 0);  -- Senal de comparaciones(LE&GE);
	WE	: out std_logic;                                   -- Senal de lectura('0') y escritura('1')
	R1	: out std_logic;                                   -- Senal registro 1
	R2	: out std_logic;                                   -- Senal registro 2
	OPC1: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Renglones)
	OPC2: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Columanas)
	FADD: out std_logic;                                   -- Bandera de direccion proceso uno
	FCOM: out std_logic_vector(1 downto 0);-- Bandera indicadora de proceso actual
	FIN	: out std_logic                                            -- Senal de fin de proceso
	);
end component;

component contador_multifuncional
	generic(
```

```
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

component registro
        generic(
        n: integer:=16
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        H       : in std_logic;
        D       : in std_logic_vector(n-1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

signal OPC1,OPC2: std_logic_vector(1 downto 0);
signal CONTADOR1,CONTADOR2: std_logic_vector(19 downto 0);
signal DATMAYOR: std_logic_vector(7 downto 0);
signal G1: std_logic;
signal DAT1,DAT2: std_logic_vector(7 downto 0);
signal R1,R2: std_logic;
signal FADD: std_logic;
signal FC1,FR1: std_logic;
signal COMP,FCOM: std_logic_vector(1 downto 0);
signal SUMARENGLON,RESTACOLUMNA: std_logic_vector(19 downto 0);
signal ADD1,ADD2,ADD3,ADD4: std_logic_vector(19 downto 0);
begin
        --
**************************************************************************************
*************************
        -- FSM


        --
**************************************************************************************
*************************
        bloque_33: FSM_derecha_izquierda_nueva port
map(RST,CLK,STT,FC1,FR1,COMP,WE,R1,R2,OPC1,OPC2,FADD,FCOM,FIN);
        --
**************************************************************************************
*************************
        -- Contadores
```

```
	--
**************************************************************************************
************************
	bloque_00: contador_multifuncional generic map(20) port map(RST,CLK,OPC1,CONTADOR1);
	bloque_01: contador_multifuncional generic map(20) port map(RST,CLK,OPC2,CONTADOR2); --
Comienza en '1'
	--
**************************************************************************************
************************
	--		Derecha_Izquierda
	--
**************************************************************************************
************************
	SUMARENGLON  <= std_logic_vector(unsigned(R)+1); -- RE1
	FC1 <= '0' when CONTADOR2(9 downto 0) < C(9 downto 0) else '1';
	FR1 <= '0' when CONTADOR1(9 downto 0) < SUMARENGLON(9 downto 0) else '1';
	ADD4 <= std_logic_vector(unsigned(CONTADOR1(9 downto 0))*unsigned(C(9 downto 0)));
	ADD3 <= std_logic_vector(unsigned(ADD4)+unsigned(DIR));
	ADD1 <= std_logic_vector(unsigned(ADD3)-unsigned(CONTADOR2));
	ADD2 <= std_logic_vector(unsigned(ADD1)-1);
	ADD <= ADD1 when FADD='0' else ADD2;
	--
**************************************************************************************
************************
	-- Datos de salida
	--
**************************************************************************************
************************
	bloque_04: registro generic map(8) port map(RST,CLK,R1,DATI(7 downto 0),DAT1);
	bloque_05: registro generic map(8) port map(RST,CLK,R2,DATI(7 downto 0),DAT2);
	G1 <= '1' when DAT1 > DAT2 else '0';
	COMP <= '0'&G1;
	DATMAYOR <= DAT1 when (FCOM(0)='1') else DAT2;
	DATO <= "00000000"&DATMAYOR;
end procesos;
```

Abajo Arriba.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity abajo_arriba_nueva is
	port(
	RST	: in std_logic;                                                    -- Senal de reset
	CLK	: in std_logic;                                                    -- Senal de reloj
	STT	: in std_logic;                                                    -- Senal de inicio de
procesamiento
	DIR	: in std_logic_vector(19 downto 0);          -- Direccion de inicio de
manipualcion de imagen(Renglon*Columna)
```

```
        R       : in std_logic_vector(19 downto 0);             -- Numero de Renglones
        C       : in std_logic_vector(19 downto 0);             -- Numero de Columnas
        DATI: in std_logic_vector(15 downto 0);         -- Dato de entrada
        DATO: out std_logic_vector(15 downto 0);        -- Dato de salida
        WE      : out std_logic;                                        -- Senal de lectura('0') y
escritura('1')
        ADD     : out std_logic_vector(19 downto 0);    -- Direccion donde se desea leer o escribir
        FIN     : out std_logic                                         -- Senal de fin de
procesamiento
        );
end abajo_arriba_nueva;

architecture procesos of abajo_arriba_nueva is

component FSM_abajo_arriba_nueva
        port(
        RST     : in std_logic;                                 -- Senal de reset
        CLK     : in std_logic;                                 -- Senal de reloj
        STT     : in std_logic;                                 -- Senal de inicio de proceso
        FC1 : in std_logic;                                     -- Senal de bandera
columnas 1
        FR1 : in std_logic;                                     -- Senal de bandera
renglones 1
        COMP: in std_logic_vector(1 downto 0);  -- Senal de comparaciones(LE&GE);
        WE      : out std_logic;                                -- Senal de lectura('0') y escritura('1')
        R1      : out std_logic;                                -- Senal registro 1
        R2      : out std_logic;                                -- Senal registro 2
        OPC1: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Renglones)
        OPC2: out std_logic_vector(1 downto 0); -- Opcion contador multifuncional(Columanas)
        FADD: out std_logic;                                    -- Bandera de direccion proceso uno
        FCOM: out std_logic_vector(1 downto 0);-- Bandera indicadora de proceso actual
        FIN     : out std_logic                                 -- Senal de fin de proceso
        );
end component;

component contador_multifuncional
        generic(
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

component registro
        generic(
        n: integer:=16
```

```
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        H       : in std_logic;
        D       : in std_logic_vector(n-1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;

signal OPC1,OPC2: std_logic_vector(1 downto 0);
signal CONTADOR1,CONTADOR2: std_logic_vector(19 downto 0);
signal DATMAYOR: std_logic_vector(7 downto 0);
signal G1: std_logic;
signal DAT1,DAT2: std_logic_vector(7 downto 0);
signal R1,R2: std_logic;
signal FADD: std_logic;
signal FC1,FR1: std_logic;
signal COMP,FCOM: std_logic_vector(1 downto 0);
signal ADD1,ADD2,ADD3,ADD4,ADD5: std_logic_vector(19 downto 0);
begin
        --
************************************************************************************
************************
        -- FSM


        --
************************************************************************************
************************
        bloque_33: FSM_abajo_arriba_nueva port
map(RST,CLK,STT,FC1,FR1,COMP,WE,R1,R2,OPC1,OPC2,FADD,FCOM,FIN);
        --
************************************************************************************
************************
        -- Contadores
        --
************************************************************************************
************************
        bloque_00: contador_multifuncional generic map(20) port map(RST,CLK,OPC1,CONTADOR1);
        bloque_01: contador_multifuncional generic map(20) port map(RST,CLK,OPC2,CONTADOR2);
        --
************************************************************************************
************************
        --      Derecha_Izquierda
        --
************************************************************************************
************************
        FR1 <= '0' when CONTADOR1(9 downto 0) < R(9 downto 0) else '1';
        FC1 <= '0' when CONTADOR2(9 downto 0) < C(9 downto 0) else '1';
```

```
        ADD5 <= std_logic_vector(unsigned(R)-unsigned(CONTADOR1));
        ADD4 <= std_logic_vector(unsigned(ADD5(9 downto 0))*unsigned(C(9 downto 0)));
        ADD3 <= std_logic_vector(unsigned(ADD4)+unsigned(DIR));
        ADD1 <= std_logic_vector(unsigned(ADD3)+unsigned(CONTADOR2));
        ADD2 <= std_logic_vector(unsigned(ADD1)-unsigned(C));
        ADD <= ADD1 when FADD='0' else ADD2;
        --
************************************************************************************
**************************
        -- Datos de salida
        --
************************************************************************************
**************************
        bloque_04: registro generic map(8) port map(RST,CLK,R1,DATI(7 downto 0),DAT1);
        bloque_05: registro generic map(8) port map(RST,CLK,R2,DATI(7 downto 0),DAT2);
        G1 <= '1' when DAT1 > DAT2 else '0';
        COMP <= '0'&G1;
        DATMAYOR <= DAT1 when (FCOM(0)='1') else DAT2;
        DATO <= "00000000"&DATMAYOR;
end procesos;
```

FSM control 4

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm_control_4 is
port(
        RST     : in std_logic;
        CLK     : in std_logic;
        STT     : in std_logic;
        STT1    : out std_logic;
        STT2    : out std_logic;
        STT3    : out std_logic;
        STT4    : out std_logic;
        FIN1    : in std_logic;
        FIN2    : in std_logic;
        FIN3    : in std_logic;
        FIN4    : in std_logic;
        FIN     : out std_logic;
        FDAT    : out std_logic_vector(2 downto 0)
);
end fsm_control_4;

architecture proceso_fsmcon of fsm_control_4 is

signal Qn,Qp: std_logic_vector(4 downto 0);

begin
```

```
Qp <= (others => '0') when (RST='0') else Qn when rising_edge(CLK);

combinacional:process(Qp,STT,FIN1,FIN2,FIN3,FIN4)
begin
case Qp is
        when "00000" =>                 -- STT PRINCIPAL
        if(STT='0')then
                Qn <= Qp;
        else
                Qn <= "00001";
        end if;
        STT1 <= '0';
        STT2 <= '0';
        STT3 <= '0';
        STT4 <= '0';
        FIN  <= '0';
        FDAT <= "000";

        when "00001" =>                 -- STT1
        STT1 <= '1';
        STT2 <= '0';
        STT3 <= '0';
        STT4 <= '0';
        FIN  <= '0';
        FDAT <= "000";
        Qn   <= "00010";

        when "00010" =>                 -- FIN1
        if(FIN1='1')then
                Qn <= "00011";
        else
                Qn <= Qp;
        end if;
        STT1 <= '0';
        STT2 <= '0';
        STT3 <= '0';
        STT4 <= '0';
        FIN  <= '0';
        FDAT <= "000";

        when "00011" =>                 -- STT2
        STT1 <= '0';
        STT2 <= '1';
        STT3 <= '0';
        STT4 <= '0';
        FIN  <= '0';
        FDAT <= "001";
        Qn          <= "00100";

        when "00100" =>                 -- FIN2
```

```
if(FIN2='1')then
        Qn <= "00101";
else
        Qn <= Qp;
end if;
STT1 <= '0';
STT2 <= '0';
STT3 <= '0';
STT4 <= '0';
FIN  <= '0';
FDAT <= "001";

when "00101" =>                 -- STT3
STT1 <= '0';
STT2 <= '0';
STT3 <= '1';
STT4 <= '0';
FIN  <= '0';
FDAT <= "010";
Qn       <= "00110";

when "00110" =>                 -- FIN3
if(FIN3='1')then
        Qn <= "00111";
else
        Qn <= Qp;
end if;
STT1 <= '0';
STT2 <= '0';
STT3 <= '0';
STT4 <= '0';
FIN  <= '0';
FDAT <= "010";

when "00111" =>                 -- STT4
STT1 <= '0';
STT2 <= '0';
STT3 <= '0';
STT4 <= '1';
FIN  <= '0';
FDAT <= "011";
Qn       <= "01000";

when "01000" =>                 -- FIN4
if(FIN4='1')then
        Qn <= "01001";
else
        Qn <= Qp;
end if;
STT1 <= '0';
```

```
                STT2 <= '0';
                STT3 <= '0';
                STT4 <= '0';
                FIN  <= '0';
                FDAT <= "011";

                when  others => -- FIN
                STT1 <= '0';
                STT2 <= '0';
                STT3 <= '0';
                STT4 <= '0';
                FIN  <= '1';
                FDAT <= "000";
                Qn   <= "00000";

        end case;

        end process combinacional;


end proceso_fsmcon;
```

Contar Pixeles.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity contar_pixeles is
        port(
        RST                 : in std_logic;
        CLK                 : in std_logic;
        STT                 : in std_logic;
        R                   : in std_logic_vector(19 downto 0);
        C                   : in std_logic_vector(19 downto 0);
        DIRECCION: in std_logic_vector(19 downto 0);
        WE                  : out std_logic;
        DATI      : in std_logic_vector(15 downto 0);
        ADD                 : out std_logic_vector(19 downto 0);
        PIXELES   : out std_logic_vector(19 downto 0);
        FIN           : out std_logic
        );
end contar_pixeles;

architecture proceso of contar_pixeles is

component FSM_contar_pixeles_negros
        port(
        RST       : in std_logic;
        CLK       : in std_logic;
        STT       : in std_logic;
```

```
        OPC1: out std_logic_vector(1 downto 0);
        OPC2: out std_logic_vector(1 downto 0);
        R1      : out std_logic;
        R2      : out std_logic;
        FRC     : in std_logic;
        L       : in std_logic;
        WE      : out std_logic;
        FIN     : out std_logic
        );
end component;

component contador_multifuncional
        generic(
        n: integer:= 20
        );
        port(
        RST     : in std_logic;
        CLK     : in std_logic;
        OPC     : in std_logic_vector(1 downto 0);
        Q       : out std_logic_vector(n-1 downto 0)
        );
end component;


signal CONTADOR1,CONTADOR2,RC: std_logic_vector(19 downto 0);
signal R1,R2,L,FRC: std_logic;
signal DAT1: std_logic_vector(7 downto 0);
signal OPC1,OPC2: std_logic_vector(1 downto 0);

begin
        RC <= std_logic_vector(unsigned(R(9 downto 0))*unsigned(C(9 downto 0)));
        bloque_00: FSM_contar_pixeles_negros port map(RST,CLK,STT,OPC1,OPC2,R1,R2,FRC,L,WE,FIN);
        bloque_01: contador_multifuncional generic map(20) port map(RST,CLK,OPC1,CONTADOR1); --
Numero de pixeles
        bloque_02: contador_multifuncional generic map(20) port map(RST,CLK,OPC2,CONTADOR2); -- Fin
de barrido a imagen
        DAT1 <= (others=>'0') when (RST='0') else DATI(7 downto 0) when rising_edge(CLK) AND (R1='1');
        L <= '1' when DAT1 < "11111111" else '0';
        FRC <= '0' when CONTADOR2 < RC else '1';
        PIXELES <= (others=>'0') when (RST='0') else CONTADOR1 when rising_edge(CLK) AND (R2='1');
        ADD <= std_logic_vector(unsigned(CONTADOR2)+unsigned(DIRECCION));
end proceso;
```

Escribir.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Escribir_3_datos is
	port(
	RST 				: in std_logic;
	CLK 				: in std_logic;
	STT 				: in std_logic;
	DIRECCION 		: in std_logic_vector(19 downto 0);
	DATI 				: in std_logic_vector(19 downto 0);
	CONTADOR_AUX : in std_logic_vector(19 downto 0);
	DAT 				: out std_logic_vector(7 downto 0);
	WE 					: out std_logic;
	ADD 				: out std_logic_vector(19 downto 0);
	FIN 				: out std_logic
	);
end Escribir_3_datos;

architecture proceso of Escribir_3_datos is

component FSM_escribir3
	port(
	RST	: in std_logic;
	CLK	: in std_logic;
	STT	: in std_logic;
	R1	: out std_logic;
	WE	: out std_logic;
	OPC : out std_logic_vector(1 downto 0);
	AUX	: out std_logic_vector(1 downto 0);
	FIN	: out std_logic
	);
end component;

component contador_multifuncional
	generic(
	n: integer:= 20
	);
	port(
	RST	: in std_logic;
	CLK	: in std_logic;
	OPC	: in std_logic_vector(1 downto 0);
	Q	: out std_logic_vector(n-1 downto 0)
	);
end component;

signal R1: std_logic;
signal OPC,AUX: std_logic_vector(1 downto 0);
signal CONTADOR1: std_logic_vector(1 downto 0);
signal D1,D2,D3: std_logic_vector(7 downto 0);
```

```
begin
	bloque_00: FSM_escribir3 port map(RST,CLK,STT,R1,WE,OPC,AUX,FIN);
	bloque_01: contador_multifuncional generic map(2) port map(RST,CLK,OPC,CONTADOR1);
	-- Direcciones
	ADD <=
std_logic_vector(unsigned(DIRECCION)+unsigned(CONTADOR_AUX)+unsigned(CONTADOR1));--
2*RECO+MEMORIA+[1 2 3]
	-- Datos
	D1 <= "0000"&DATI(19 downto 16);
	D2 <= DATI(15 downto 8);
	D3 <= DATI(7 downto 0);
	DAT <= D1 when AUX="00" else D2 when AUX="01" else D3;
end proceso;
```