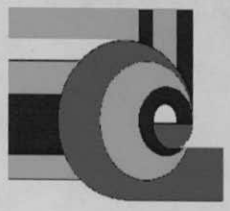
INSTITUTO POLITECNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN



Arquitecturas específicas para la implementación de primitivas morfológicas

T E S I S

que para obtener el grado de:

MAESTRO EN INGENIERÍA DE CÓMPUTO

CON ESPECIALIDAD EN SISTEMAS DIGITALES

Presenta

AGUSTÍN CRUZ CONTRERAS

Director de Tesis: Dr. Juan Luis Díaz de León Santiago

Codirector: Dr. Oleksiy Pogrebnyak

México, D.F. a 25 de mayo del 2006

SIP-14

INSTITUTO POLITECNICO NACIONAL
SECRETARIA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 18:00 horas del día 6 del mes de Abril de 2006 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis de grado titulada:

"ARQUITECTURAS ESPECÍFICAS PARA LA IMPLEMENTACIÓN DE PRIMITIVAS MORFOLÓGICAS"

CRUZ	CONTRERAS	AGUSTÍN
Apellido paterno	materno	nombre(s)

Con registro: A 9 3 0 9 7 2

aspirante al grado de: ***MAESTRO EN CIENCIAS EN INGENIERIA DE CÓMPUTO CON ESPECIALIDAD EN SISTEMAS DIGITALES***

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACIÓN DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Presidente

Dr. Sergio Suárez Guerra

Secretario

Dr. Luis Rastor Sánchez Fernández

Primer vocal
(Director de tesis)

Dr. Juan Luis Díaz de León Santiago

Segundo vocal
(Co-director)

Dr. Oleksiy Pogrebnyak

Tercer Vocal

Dr. Carlos Fernando Aguilar Ibañez

Suplente

M en C. Pablo Manrique Ramírez

EL PRESIDENTE DEL COLEGIO

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
DIRECCION

Dr. Hugo César Castellanos

INSTITUTO POLITECNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESION DE DERECHOS

En la Ciudad de México el día 25 del mes de mayo del año 2006, el que suscribe: Agustín Cruz Contreras, alumno del Programa de Maestría en Ingeniería de Cómputo, con número de registro A930972, adscrito al Laboratorio de Inteligencia artificial, manifiesta que es autor intelectual del presente trabajo de Tesis, bajo la dirección de Dr. Juan Luis Díaz de León Santiago, y cede los derechos del trabajo intitulado "*Arquitecturas especificas para la implementación de primitivas morfológicas*", al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección acruz@ipn.mx . Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Agustín Cruz Contreras

Nombre y firma

Agradecimientos

Agradezco al Instituto Politécnico Nacional la autorización del ejercicio de año sabático; por lo que fue posible la dedicación completa a la realización de esta Tesis de Maestría.

A mi centro de adscripción CIDETEC, por el apoyo con las tarjetas de evaluación usadas en la implementación del sistema de procesamiento.

Al Director de Tesis Dr. Juan Luis Díaz de León Santiago, quien con su amplia experiencia inspiró el desarrollo de esta Tesis. Al Codirector Dr. Oleksiy Pogrebnyak, por aportar su tiempo y experiencia a la revisión y conclusión de la Tesis. Al M. en C. Pablo Manrique Ramírez, por su oportuna orientación en la realización de trámites y preparación del documento.

A mi esposa y mis pequeños David e Itzel; por ser el motor que me impulsa en cada día de vida.

Muchas gracias.

RESUMEN

En el presente trabajo de Tesis de Maestría se define e implementa una arquitectura específica para el procesamiento morfológico de imágenes progresivas de alta definición en blanco y negro.

El principio de la arquitectura se basa en un procesamiento local de la imagen en el momento que ésta fluye. Con registros de corrimiento se retienen las líneas necesarias para cubrir la vecindad requerida. Las operaciones de erosión y dilatación se obtienen a través de operadores lógicos AND y OR.

El hardware desarrollado se implementa utilizando FPGAs con una densidad de 100k compuertas y una frecuencia máxima de 200 Mhz.

La señal de entrada tiene la forma RGB con 8 bits por color, el hardware requerido para la conversión de RGB a blanco y negro se integra en el mismo FPGA.

ABSTRACT

This work of Thesis defines and implements a specific architecture for the morphological processing of progressive high-definition black and white images.

The principle for the proposed architecture is based on a local processing of the image, right at the moment that the image is flowing. Shift registers are used to retain the lines that are necessary to cover the required neighborhood. The operations of erosion and dilation are obtained through logical AND and OR operators.

The developed hardware is implemented using FPGA technology, by mean of devices with a density of 100k gates, and 200 Mhz as the maximum frequency for the system.

The input signal has a RGB format with 8 bits per color; the hardware required for the conversion of RGB to black and white representation is integrated in the same FPGA.

Contenido

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Capítulo 1: Introducción

Los métodos de procesamiento de imágenes digitales se fundamentan en dos áreas principales de aplicación: a) mejora de la calidad para la interpretación humana; b) procesamiento de los datos de la escena para la percepción por las computadoras de forma autónoma.

En el intento por dotar a las máquinas de un sistema de visión, aparece el concepto de *Visión por Computadora*. La visión por computadora es una rama de la inteligencia artificial que tiene por objetivo modelar matemáticamente los procesos de percepción visual en los seres vivos y generar programas que permitan simular estas capacidades visuales por computadora.

La información visual es una de las principales fuentes de datos del mundo real, resulta útil proveer a una computadora de visión por medio de cámaras, que junto con otros mecanismos como el aprendizaje, hagan de ésta, una herramienta capaz de detectar y ubicar objetos en el mundo real, lo cual es el objetivo principal de la Visión por Computadora.

La visión por computador también se aplica en el control de calidad de los procesos de producción. En procesos donde la calidad es un factor altamente importante, la inspección estadística no se puede considerar como un criterio efectivo, por lo que la inspección al cien por ciento es necesaria.

Actualmente existen cámaras inteligentes que cuentan con una arquitectura que les permite el análisis de datos sin la necesidad de intervención humana. Sus algoritmos de procesamiento se desarrollan con base en modelos de inteligencia artificial, lo que les permite el reconocimiento e identificación de objetos, a partir de las imágenes adquiridas.

Las cámaras inteligentes se desarrollan a la medida de las necesidades de inspección, en los procesos de fabricación estas cámaras son capaces de realizar una inspección de múltiples características en cada uno de los elementos de una línea de producción, a una velocidad tal, que las interrupciones en el proceso de fabricación son prácticamente imperceptibles.

Los objetos pueden ser localizados con alta precisión, inclusive si se encuentran en movimiento, y ante variaciones de las condiciones del entorno donde se desarrolla el proceso, tales como cambios en los niveles de iluminación.

En una cámara inteligente el procesamiento se debe realizar en tiempo real, con el propósito de dar una respuesta antes de que las condiciones del entorno hayan variado. El uso de procesadores específicos acelera en gran medida el procesamiento, y dado que estos se implementan en un chip, resulta fácil integrarlos en la misma cámara.

Una arquitectura específica para un algoritmo puede tener un rendimiento de 10 a 1000 veces superior al de una implementación en un DSP. Los algoritmos de bajo nivel en visión por computadora y análisis de imágenes, son candidatos ideales para explorar arquitecturas específicas que aceleren su rendimiento.

1.1 Antecedentes

Los procesamientos lineales surgidos a mediados de los años 50 como consecuencia de la carrera espacial, resuelven con éxito la eliminación de ruido y otras funciones importantes para las que fueron creados, pero presentan elevados tiempos de procesamiento, alto costo del hardware y una serie de limitaciones en cuanto a la degradación de las imágenes que procesan.

Como respuesta a estas limitaciones nace la *Morfología Matemática*, una teoría surgida de las investigaciones de G. Matheron y J. Serra en los años 60; fundamentada en el uso de *procesos no lineales* derivados de la comparación de la imagen con un patrón geométrico sencillo denominado *Elemento Estructurante*, y que aporta una serie de ventajas:

- Baja complejidad computacional, basándose en operaciones lógicas
- Se conservan las características geométricas de la imagen procesada
- Dos primitivas básicas: *Dilatación* y *Erosión,* con combinaciones de las mismas se construyen todos los operadores

La Morfología Matemática es una técnica de procesamiento y análisis de imágenes relativamente joven, la cual ha demostrado gran capacidad para solventar una amplia gama de problemas sobre imágenes binarias (blanco y negro) o numéricas (escala de grises o a color). La Morfología matemática está basada en geometría y forma. Las operaciones morfológicas simplifican las imágenes y preservan las formas principales de los objetos.

Las imágenes que trata la morfología matemática son muy variadas: radiografías de objetos tanto técnicos como biológicos, imágenes de microscopía electrónica o análisis de escenas para el control de vehículos. Por otra parte, la morfología matemática ha originado también técnicas de tratamiento de señales (medida de la glucosa en sangre) y técnicas originales y prometedoras para la compresión de imágenes en movimiento.

Los desarrollos dedicados específicamente al procesamiento de operadores morfológicos en tiempo real; se dan a través de soluciones basadas totalmente en hardware. El hardware se implementa por medio de circuitos integrados de aplicación específica (ASIC), en estos circuitos se implementa el respectivo algoritmo, siendo el paralelismo la alternativa más explotada por los diseñadores.

El desarrollo de circuitos ASIC requiere de habilidades y herramientas para el diseño, en la implementación del circuito se requieren los servicios de una fundidora. Las herramientas para el desarrollo de circuitos ASIC son muy costosas, por lo que pocas instituciones tienen acceso a esta tecnología.

Entre los trabajos publicados en los cuales se abordan específicamente alternativas para el procesamiento de morfología en tiempo real, podemos encontrar los siguientes:

"An Asynchronous 16*16 Pixel Array-Processor for Morphological Filtering of Greyscale Images" [12]. En este trabajo se presenta un chip VLSI que consiste en un arreglo de procesamiento de 16 x 16. El chip se fabrica usando CNET/SGS-Thomson 0.55 μm CMOS, tecnología de tres capas metálicas. Incluye 800,000 transistores en un área de 8 x 9 mm^2. El desarrollo permite un filtrado morfológico interactivo de niveles de gris en imágenes de 256 x 256 pixeles a 30 cuadros por segundo usando un solo chip.

"A Monolithic Mixed-Mode Implementation of Sum-of-Product Arrays for Performing Binary Morphological Image Processing" [13]. En este trabajo se presenta el desarrollo de un chip monolítico en el cual se implementa un arreglo de suma de productos con el propósito de realizar procesamiento morfológico de imágenes. El chip se fabrica usando un

proceso doble-poly ORBIT CMOS 2.0 μ de doble-metal, sus dimensiones son 100μm x100 μm. El chip procesa las operaciones de erosión y dilatación con un elemento estructurante de 3 x 3.

"An Analogue VLSI Morphological Image Processing Circuit" [5]. En este trabajo se describe el desarrollo de un Circuito VLSI para el procesamiento morfológico de imágenes, el circuito se fabrica usando un proceso estándar 2.0μm CMOS, con un tamaño de 187μm x 187μm. En éste se explota el paralelismo de los operadores del procesamiento morfológico a través de arreglos de arquitecturas paralelas VLSI. El circuito procesa la operación de dilatación en tonos de gris.

En los trabajos anteriores el chip que se desarrolla permite procesar los operadores de de la morfología en forma singular, esto quiere decir, que si deseamos implementar un algoritmo donde se usan varios operadores morfológicos, se requerirá de varias chips y el circuito impreso para su montaje e interconexión, lo anterior implica tener que desarrollar un circuito impreso para cada algoritmo que se quiera implementar.

En los trabajos presentados no se menciona de donde proviene la imagen por procesar, ni a donde será enviada después de su procesamiento. Considerando que se trata de procesamiento en tiempo real, la señal de entrada debe ser una señal de video, por lo tanto también se requiere de hardware para adecuar y sincronizar la señal de video con el chip de procesamiento.

1.2 Motivación

El procesamiento de imágenes en tiempo real está motivado por diversas aplicaciones; entre las más comunes podemos mencionar: inspección industrial, control de tráfico, control de vehículos, visión robótica, vigilancia, sistemas de seguridad e imágenes médicas. Estas aplicaciones tienen en común la presencia de entornos dinámicos, por el carácter cambiante de este entorno, el sistema debe tomar una decisión y aplicarla antes de que varíen las condiciones del mismo.

En el intento por cubrir necesidades de este tipo surgen desarrollos basados en arquitecturas de tipo paralelo como son: Procesadores Digitales de Señales (DSPs), PC-clusters (varias computadoras conectadas en paralelo); Sistemas de Instrucción única para Múltiples Datos

(SIMD), en estos una única unidad de control genera las instrucciones a diferentes unidades de procesamiento, como ejemplo tenemos: 3DNow! de AMD y SSE (Streaming SIMD Extensions) de Intel.

El paralelismo ofrece muchas ventajas pero, a pesar de esto, la implementación de los algoritmos de procesamiento no se puede dar en un paso. Ante la necesidad de una respuesta inmediata por parte de los sistemas de visión se han desarrollado diversas alternativas: las arquitecturas específicas plantean una implementación del algoritmo a la medida esto es; una solución cien por ciento hardware que proporcione recursos dedicados para la ejecución del algoritmo. Las arquitecturas específicas se pueden implementa a través de circuitos integrados de aplicación específica (ASIC), y en caso de que la complejidad del algoritmo lo permita se pueden utilizar FPGAs.

1.3 Objetivos

General:

Definir e implementar una arquitectura específica para procesamiento morfológico de imágenes en blanco y negro, con una resolución de 640 x 480 a 60 cuadros por segundo.

Específicos:

- Convertir de serie a paralelo la señal de video. La conversión se realiza a través de chips decodificadores como el SIL161BCT100 de Silicon Image.

- Binarizar la señal de video en formato RGB proveniente de la tarjeta de decodificación. El hardware requerido en este proceso se toma del FPGA.

- Formar registros de corrimiento de 640 bits a través de las LUTs del FPGA.

- Integrar con los registros de corrimiento la arquitectura propuesta, formando macros para las operaciones de erosión y dilatación. Por medio de la combinación de estos dos operadores se implementa cualquier algoritmo de la Morfología Matemática.

- Adecuar la señal en B&N y procesada para su despliegue a través de monitores analógicos. Para los monitores analógicos los niveles de la señales deben estar entre 0V y 1V, el acoplamiento de impedancia es a 75 ohms.

- Implementar con las macros desarrolladas el algoritmo para la extracción de contornos.

1.4 Aportaciones de la Tesis

Una aportación de esta Tesis consiste en el trabajo de conceptuar e implementar la Morfología Matemática en forma binaria esto es; asociar las operaciones de la morfología matemática (erosión y dilatación) con elementos natos de hardware como son: registros de corrimiento y compuertas lógicas.

Procesar la imagen en forma de señal es otra aportación de la Tesis; el hecho de procesar la imagen en forma de señal permite eliminar los tiempos de captura y acceso a memoria. Al trabajar con imágenes en memoria principal se requiere el acceso a ésta para lectura o escritura. Considerando que en el mejor de los casos se puede tener un acceso a 32 bits (computadoras de 32 bits) en cada operación de lectura o escritura, en una imagen de 640 x 480 se requiere de más de 300,000 accesos a memoria para escribirla o leerla. Estos tiempos de acceso se adicionan al tiempo de procesamiento, siendo en la mayoría de los casos, mayor el tiempo de acceso a memoria, que el tiempo de procesamiento. Lo anterior se debe principalmente a la arquitectura de las computadoras, en éstas el procesador trabaja a mayor velocidad en su núcleo que el resto del sistema, motivo por el cual las tareas dentro del micro se ejecutan a mayor velocidad en comparación con tareas externas.

Otra aportación de no menos consideración es el entorno de desarrollo integrado con las macros desarrolladas y el editor de esquemas. Con este entorno se tiene una plataforma flexible y versátil, la cual permite construir diversos operadores para cuestiones didácticas y aplicaciones de la Morfología Matemática.

1.5 Estructura

En el segundo capítulo se presenta el estado del arte con relación al procesamiento de imágenes en tiempo real, se revisan diversos trabajos; procesamiento paralelo y arquitecturas específicas son las tendencias más representativas en esta área.

En el tercer capítulo se presenta una descripción detallada de los conceptos fundamentales de la Morfología Matemática: definiciones de conjuntos y operaciones básicas para imágenes en blanco y negro. La idea para la implementación de la arquitectura propuesta en este trabajo de tesis se basa en el tercer método para la obtención de la erosión y dilatación, descrito en este capítulo.

En el cuarto capítulo se revisan las características del puerto digital de video (DVI) y los FPGAs de la familia Spartan II. A través del FPGA se implementa la arquitectura propuesta, a partir del puerto DVI se decodifican las señales de video RGB y los pulsos de sincronía.

En el capítulo cinco se describe la arquitectura propuesta en este trabajo, se explica a detalle la construcción de las macros de erosión y dilatación realizadas con base en esta arquitectura. Se presentan ejemplos de aplicación a través de la operación para la extracción del contorno.

En el capítulo seis se redactan las conclusiones y se comentan las posibles opciones para un trabajo futuro.

Capítulo 2

Estado del arte

Capítulo 2: Estado del arte

La mayoría de los algoritmos para el procesamiento de imágenes presentan un alto costo computacional, esta situación limita su aplicación en sistemas de visión con entorno dinámico, donde se requiere procesamiento en tiempo real. En el intento por reducir los tiempos de procesamiento se han desarrollado diversas alternativas, en común todas estas alternativas han tendido que recurrir al aumento del hardware, desarrollando arquitecturas y lenguajes para el procesamiento en paralelo de sus algoritmos. A continuación se abordan algunas de las alternativas desarrolladas para el procesamiento de imágenes en tiempo real.

2.1 Lenguajes de procesamiento paralelo

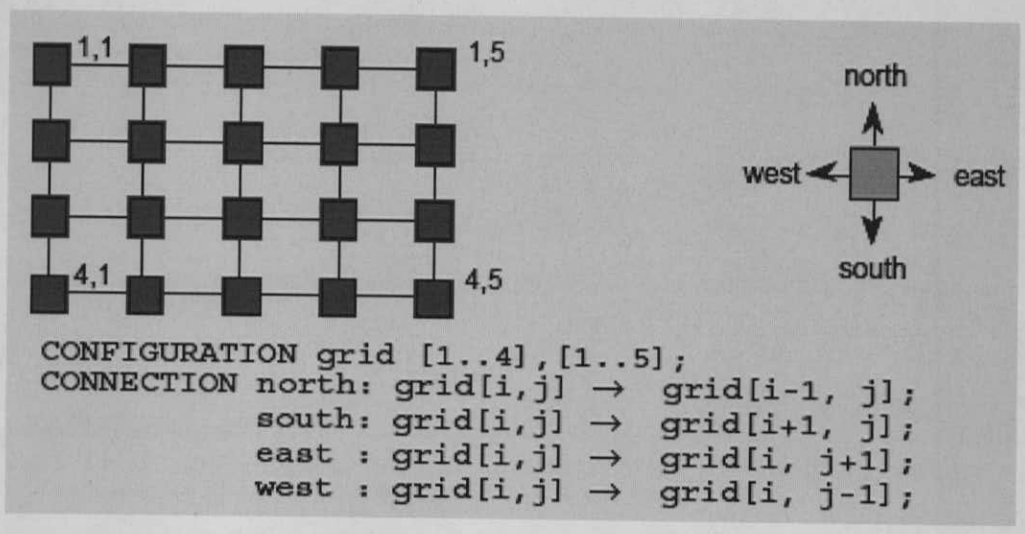
2.1.1 Parallaxis

Parallaxis es un lenguaje para programación [2] en paralelo independiente de la máquina, trabaja a través de arquitecturas de simple instrucción múltiple dato (SIMD). Se basa en el lenguaje de programación secuencial Modula-2, la cantidad de procesadores y conexión entre estos se da en forma virtual, y es determinada por el programador de la aplicación.

El entorno de simulación permite realizar el estudio de los fundamentos del procesamiento paralelo, usando una computadora normal. Se pueden desarrollar programas paralelos que posteriormente se ejecuten en sistemas de cómputo paralelo. El entorno de programación de Parallaxis está disponible como software de dominio público.

Con la palabra reservada CONFIGURATION, se define el tipo de topología y el número de elementos de procesamiento (PE), esto es análogo a declarar un arreglo. Con la palabra reservada CONNECTION, se define como se conecta el procesador con su alrededor.

La figura 2.1 presenta la declaración de las topologías de retícula, árbol y cubo. Se muestra la declaración de la topología y la interconexión entre cada procesador y su alrededor.



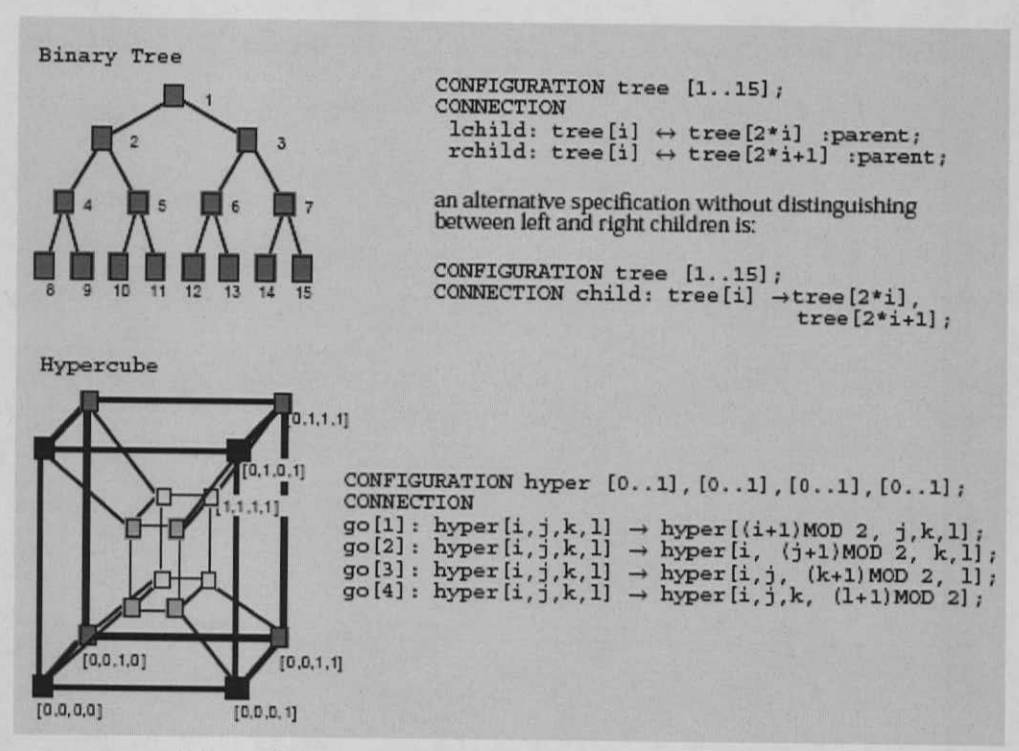


Figura 2.1: Topología de retícula, árbol e hypercubo

En la declaración de datos, Parallaxis distingue entre variables de tipo escalar y tipo vectorial. Los datos escalares son puestos en el procesador de control, los datos vectoriales son distribuidos a los PEs creados en forma virtual. La figura 2.2 presenta un gráfico de la distribución de los datos según su tipo.

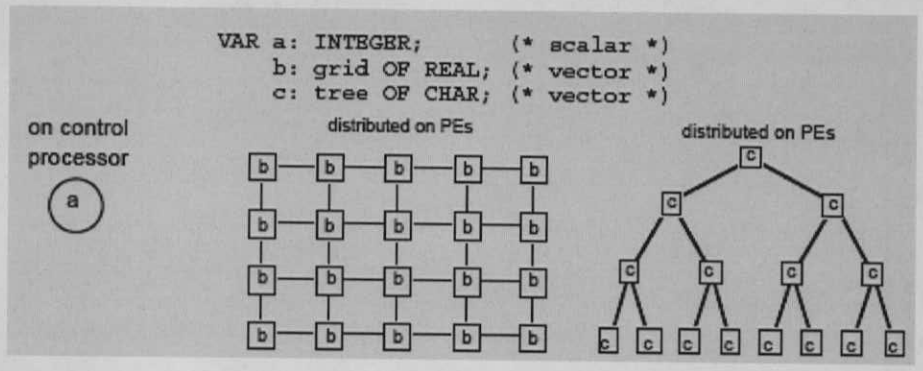


Figura 2.2: Ubicación de datos escalares y vectoriales

Las instrucciones de entrada-salida en Parallaxis son muy similares a las de Modula-2, en Parallaxis las instrucciones read o write pueden tener argumentos escalares o vectoriales. Con el propósito de hacer legible el despliegue en una salida vectorial, se agregan espacios y líneas en blanco. A continuación se presenta una lista con las instrucciones de entrada-salida.

WriteLn; Inicia nueva línea
Write(c); Escribe un caracter c
WriteString(s) Escribe una cadena s
WriteInt(i,l); Escribe un entero i usando l espacios como mínimo
WriteCard(c,l); Escribe un cardinal c usando l espacios como mínimo
WriteReal(r,l); Escribe un real r usando l espacios de impresión
WriteFixPt(r,l,m); Escribe un real r usando l espacios como mínimo y m espacios para dec.
WriteBool(b,l); Escribe un boolean b usando l espacios como mínimo
Read(c); Lee un carácter c
ReadString(s) Lee una cadena s
ReadInt(i); Lee un entero i
ReadCard(c); Lee un cardinal c
ReadReal(r); Lee un real r
ReadBool(b); Lee un boolean b
OpenOutput(s); Abre un archive de nombre s para escritura
CloseOutput; Cierra archivo
OpenInput(s); Abre archivo de nombre s para lectura
CloseInput; Cierra archivo

Las estructuras de control en Parallaxis son idénticas a las de Modula-2, al igual que en Pascal las estructuras terminan en END a excepción del REPEAT. Todas las estructuras de control pueden ser usadas con argumentos escalares o vectoriales. En la figuras 2.3 se muestran las estructuras de control existentes en Parallaxis.

```
IF x=0 THEN y:=1; z:=5
       ELSE y:=2
END;

FOR x:=1 TO 10 DO
  y:=2*y
END;

WHILE x>0 DO
  x:=x-1; y:=2*y
END;

REPEAT
  x:=x DIV 2
UNTIL x<7;
```

Figura 2.3: Estructuras de control en Parallaxis

Parallaxis se puede utilizar en diversas aplicaciones como son: Generación de fractales, Autómatas celulares, Ordenamiento, Procesamiento de imágenes, Simulación. De las aplicaciones anteriores, es de nuestro interés el procesamiento de imágenes. A continuación se hará una breve presentación de aspectos involucrados en el procesamiento de imágenes, considerando la implementación de operadores morfológicos.

Para representar una imagen en Parallaxis se considera un arreglo de PEs de dos dimensiones. En este arreglo cada PE corresponde a un pixel de la imagen. La figura 2.4 corresponde a la declaración de la topología usada para representar una imagen.

```
1 CONFIGURATION grid[*],[*];
2 CONNECTION
3 right:grid[i,j]<->grid[i  ,j+1]:left;
4 up   :grid[i,j]<->grid[i-1,j  ]:down;
5 up_l :grid[i,j]<->grid[i-1,j-1]:down_r;
6 up_r :grid[i,j]<->grid[i-1,j+1]:down_l;
```

Figura 2.4: Declaración de la topología para representar una imagen

En la figura 1.5 se presenta la forma en la que se organizan los procesadores conforme a la declaración de la topología presentada en la figura 2.4.

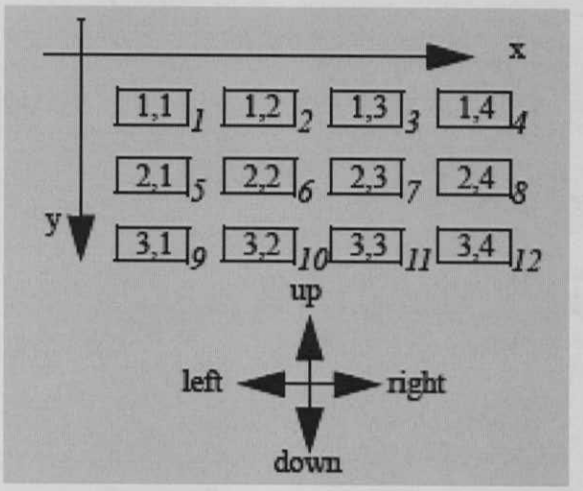


Figura 2.5: Organización de PEs para representar una imagen

El tipo de datos en la representación paralela de una imagen, distingue entre una imagen en blanco y negro, tonos de gris y color. En la figura 2.6 se presenta la declaración de datos para cada uno de estos casos.

```
 1  TYPE binary = BOOLEAN;
 2       gray   = [0..255];
 3       color  = RECORD
 4                  red, green, blue: gray
 5                END;
 6
 7  CONST b_black =  TRUE;
 8        b_white = FALSE;
 9        g_black =     0;
10        g_white =   255;
11        c_black = color(  0,   0,   0);
12        c_white = color(255,255,255);
```

Figura 2.6: Declaración de datos para imágenes en color, tonos de gris y b&n

Los operadores morfológicos básicos: erosión y dilatación se pueden implementar a través de Parallaxis [1]. En la figura 2.7 se presentan los procedimientos correspondientes para los

operadores de erosión y dilatación, aplicados en imágenes en blanco y negro. Estos procedimientos consisten en aplicar la AND entre el punto a procesar y su vecindad; para obtener la erosión de la imagen, y la OR para obtener la dilatación. En la figura 2.8 se muestran imágenes en blanco y negro correspondientes a las operaciones de erosión y dilatación.

```
 1 PROCEDURE erosion(
 2           img: grid OF binary):
 3           grid OF binary;
 4 VAR res: grid OF binary;
 5 BEGIN
 6   res := img AND MOVE.left(img)
 7              AND MOVE.right(img);
 8   res := res AND MOVE.up(res)
 9              AND MOVE.down(res);
10   RETURN res
11 END erosion;
```

```
 1 PROCEDURE dilation(
 2           img: grid OF binary):
 3           grid OF binary;
 4 VAR res: grid OF binary;
 5 BEGIN
 6   res := img OR MOVE.left(img)
 7              OR MOVE.right(img);
 8   res := res OR MOVE.up(res)
 9              OR MOVE.down(res);
10   RETURN res
11 END dilation;
```

Figura 2.7: Procedimientos para la obtención de la erosión y dilatación en imágenes en b&n

Los operadores de apertura y cerradura se obtienen a partir de combinaciones de los operadores básicos de erosión y dilatación. Para la cerradura se aplica primero la dilatación y posteriormente la erosión. Para la apertura se aplica la erosión y posteriormente la dilatación. En la figura 2.9 se muestran los procedimientos para los operadores de apertura y cerradura.

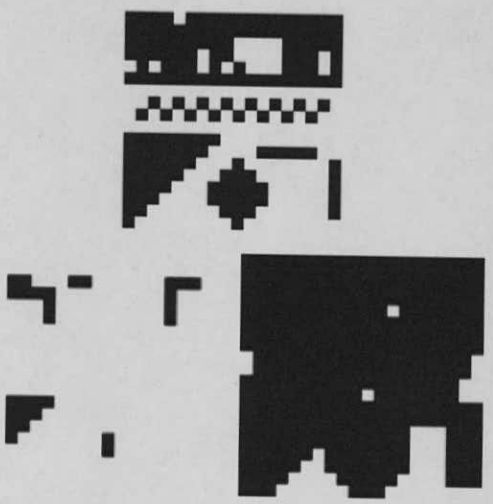


Figura 2.8: Imagen original, erosión, dilatación

```
1 PROCEDURE open(img:
2    grid OF binary): grid OF binary;
3 BEGIN
4   RETURN dilation( erosion(img) )
5 END open;
```

```
1 PROCEDURE close(img:
2    grid OF binary): grid OF binary;
3 BEGIN
4   RETURN erosion( dilation(img) )
5 END close;
```

Figura 2.9 Procedimientos para los operadores de erosión y dilatación

En la figura 2.10 se presenta una imagen y las correspondientes para los operadores de erosión, dilatación, apertura y cerradura.

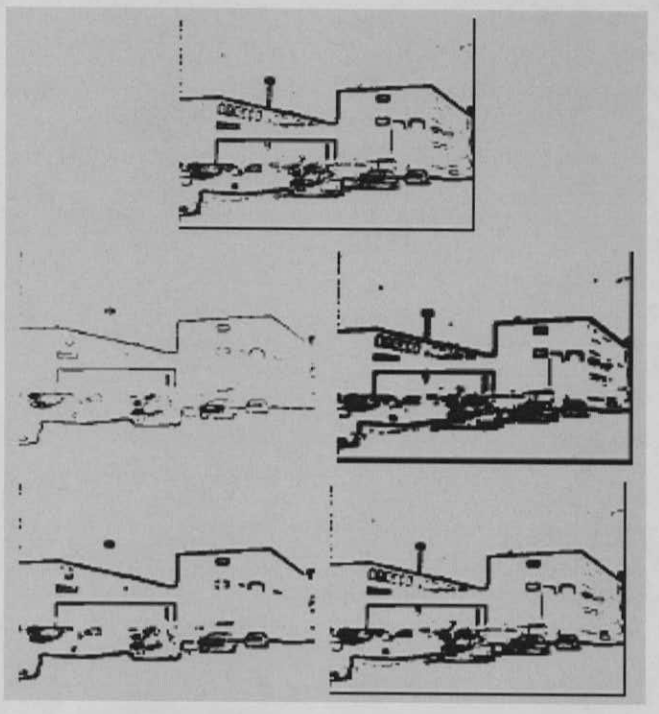


Figura 2.10 original, erosión, dilatación, apertura y cerradura

El operador de contorno es un método eficiente para encontrar la frontera interna de una imagen, el contorno se determina por la diferencia entre la imagen y su erosión. En la figura 2.11 se presenta el procedimiento para la obtención del contorno.

```
1 PROCEDURE boundary(img:
2       grid OF binary): grid OF binary;
3 BEGIN
4   RETURN img AND NOT erosion(img)
5 END boundary;
```

Figura 2.11 Procedimiento para la obtención del contorno

2.1.2 ISETL-LINDA

LINDA es un lenguaje de coordinación paralela, el cual usa espacios de tuplas compartidas para permitir la comunicación entre procesos. Es independiente de la arquitectura de la máquina, la comunicación entre procesos es asíncrona, y se puede a dar a través de espacios de tuplas. Las tuplas pueden ser puestas en un espacio y leídas o removidas de éste. Como Linda sólo se encarga de los procesos de coordinación, éste se encuentra embebido en otro lenguaje, agregando nuevas primitivas al lenguaje en cual es embebido. ISELT-LINDA es el lenguaje basado en conjuntos en el cual LINDA es embebido en ISELT.

LINDA provee cuatro operadores sobre espacios de tuplas.

- out (tuple); da un lugar a la tupla en el espacio de tuplas.
- in(template); intenta ajustar la plantilla con alguna tupla en el espacio de tuplas. Si el ajuste ocurre, la tupla es removida del espacio de tuplas, en caso contrario, se mantine en espera hasta tener una tupla disponible.
- Rd(template); es lo mismo que in excepto que la tupla ajustada no se remueve del espacio de tuplas, una copia se entrega como resultado.
- Eval(tuple); crea un llamado en tuplas activas, cada elemento de la tupla es evaluado concurrentemente, y al terminar, la tupla de resultados es colocada en el espacio de tuplas.

Los tipos principales de datos en ISETL son conjuntos y tuplas. Una tupla es una lista de objetos de tipo heterogéneo, delimitada por corchetes [ ], y separada por comas. Por ejemplo: [1, true, "hola"] es una tupla de tres objetos, cuyo tipo es entero, boolean y cadena. Se pueden describir tuplas por rango; por ejemplo: [1..5], se refiere a la tupla [1,2,3,4,5]. También se pueden describir por formación: la tupla [e(x): x en S | p(x)] es la tupla creada de los elementos de S, donde S es un conjunto o tupla, que tiene la propiedad P. Por ejemplo: [e(x) : x en [1..5] | par (X)], es la tupla [2,4]. Los conjuntos se construyen en forma similar usando llaves { }.

En ISELT-LINDA los espacios de tuplas son incorporados para proveer un nuevo tipo, llamado bag. Un bag es similar a un conjunto excepto que la repetición de elementos es permitida. De este modo la bag {|1,2,3,3,4|} es lo mismo que {| 3,4,2,3,1|}. Las operaciones

de unión e intersección no se aplican al tipo bag, la única forma de manipular el tipo bag es a través de las primitivas de LINDA.

ISELT maneja dos tipos de construcciones en sus programas: expresiones y sentencias. Las expresiones son objetos de sus principales tipos de datos: enteros de múltiple precisión, números en punto flotante y números reales. Las expresiones se construyen usando operados aritméticos tales como + y -. Varios operadores son sobrecargados por diferentes tipos, ISETL es un lenguaje poco tipado. Las sentencias incluyen condicionales, iteraciones y sentencias de impresión.

Para la asignación se utiliza ":=", la asignación se puede hacer sobre variables o sobre algún elemento de una tupla de variables. Por ejemplo Q(i):= e, se asigna "e" al elemento "i" de la tupla "Q". Se utiliza el punto y coma ";" como terminador de sentencias.

El condicional se implementa usando la estructura if ...then...else...end, la sentencia else es opcional. Los ciclos se implementan usando la estructura while...do...end.

Funciones para manipulación de espacios de tuplas:

- lout(ts,t); coloca la tupla t en el espacio ts.
- lin(ts,t); toma la plantilla t y la busca una tupla que se ajuste en el espacio ts. Si la tupla es encontrada, ésta es removida del espacio y devuelta como resultado.
- lrd(ts,t); toma la plantilla t y la busca una tupla que se ajuste en el espacio ts. Si la tupla es encontrada una copia es retornada como resultado.
- Lcollet(ts,tt,t); toma los dos espacios de tuplas, ts y tt, junto con la plantilla t y mueve las tuplas en ts que se ajustan a t a el espacio ts, retorna como resultado el número de tuplas desplazadas.

A través de ISELT-LINDA [10] se pueden implementar los operadores de la morfología matemática. La dilatación es la unión de los conjuntos de puntos producidos cuando cada pixel de la imagen es trasladado por cada pixel del elemento estructurante. La función para obtener la traslación en ISELT-LINDA se muestra en la figura 2.12.

```
translate := func(image,vec);
  local ans, member, lp2, ans2;
  ans := {};
  for member in image do
    ans := ans union {add_vec(member,vec)};
  end for;
  return ans;
end func;
```

Figura 2.12 Función escrita en ISETL para la obtención de la traslación

La función toma cada pixel del conjunto de la imagen, para producir un conjunto que contiene los pixeles de la imagen trasladados por el vector, el resultado es la unión de los conjuntos, y es retornado al término de la función. La funcion add_vec checa la cardinalidad de dos tuplas, y retorna la tupla que representa el vector adición de las dos tuplas recibidas.

La operación de dilatación se realiza al tomar cada pixel del elemento estructurante y trasladarlo con la imagen, la unión de las traslaciones corresponde al resultado. En la figura 2.13 se muestra la función para la dilatación.

```
dilate := func(image, structelement);
  local ans, member;
  ans := {};
  for member in structelement do
    ans := ans union translate(image,lp);
  end for;
  return ans;
end func;
```

Figura 2.13 Función escrita en ISETL para la obtención de la dilatación

2.2 IMAP-VISION

El sistema de procesamiento de imágenes en tiempo real IMAP-VISION [4], es un desarrollo de NEC Incubation Center. Se basa en un sistema de procesamiento de simple instrucción múltiple dato (Single Instruction, Multiple Datastream, SIMD), formado por un arreglo de procesadores con memoria integrada (Integrated Memory Array Processor, IMAP). El sistema se implementa a través de una tarjeta con bus PCI para su interconexión con PC.

Para la programación del sistema NEC desarrolla un lenguaje de alto nivel llamado 1DC (*One-dimensional C*) independiente de la máquina, y un lenguaje ensamblador. El paquete de software para IMAP-VISION, incluye driver, compilador, ligador, depurador y simulador. Soporta plataforma PC con los sistemas operativos Windows y Linux; estaciones de trabajo NEC MIPS y Sun SPARCs.

La tarjeta de IMAP-VISION (figura 2.14) contiene 256 elementos de procesamiento (PEs) de 8 bits, cada PE trabaja sobre una columna de la imagen. Todos los PE (fig.2.15) se integran con un procesador de 8 bits con 16 registros de 8 bits de propósito general y una memoria interna de 1-Kb.

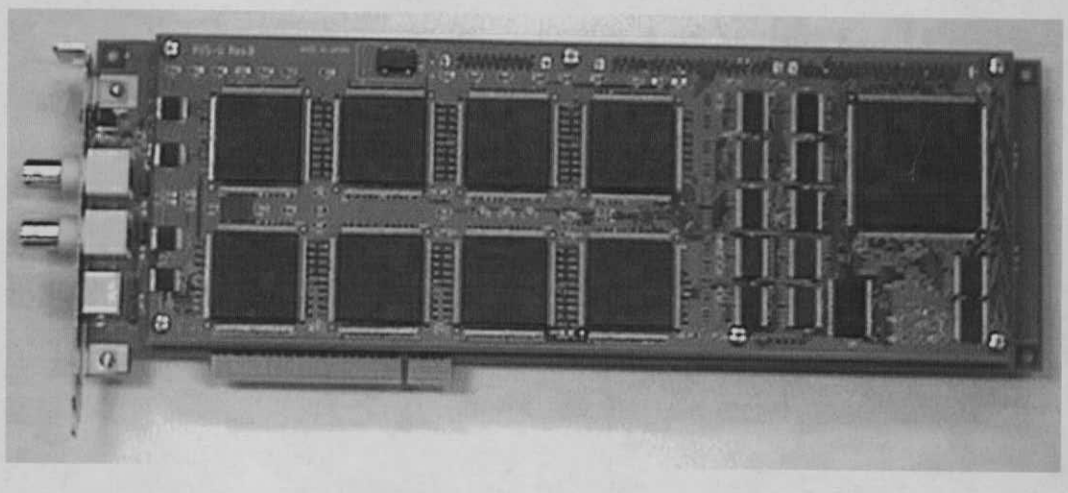


Figura 2.14: Tarjeta IMAP-VISION.

Los PE cuentan con un registro de mascara para determinar si se encuentran activos. Los PEs activos ejecutan al mismo tiempo la misma instrucción, pero estos pueden accesar diferente localidad en su memoria interna, por medio de direccionamiento indirecto.

Un chip IMAP-VISION (figura 2.15) se integra con 32 procesadores y 32 memorias síncronas de acceso aleatorio (SRAM) de 8-Kbit. Una tareta IMAP-VISON lleva ocho chips IMAP-VISION.

Aparte de la memoria interna de 1-Kb en cada elemento de procesamiento, se tiene disponible una memoria externa de 16 MB. Esta memoria no puede ser accesada directamente como memoria de trabajo por los PEs, pero se pueden almacenar datos provenientes de las memorias internas. La memoria externa es usada principalmente por el hardware de video para almacenar cuadros de video entrante o saliente, el tiempo de transferencia es de 33ms, con lo que se tiene una velocidad de 30 cuadros por segundo.

El hardware de video es incluido en la misma tarjeta. La señal proveniente del procesador de control es convertida en una imagen digital de 256x240 o 512x480, ésta se almacena en memoria interna o externa. La imagen puede ser de color o tonos de gris. El hardware de video puede leer una imagen de memoria y enviarla a la salida convertida en señal de video analógico. En la tarjeta se pueden conectar dispositivos de video compuesto, tales como: cámara de video y televisor; de igual manera se puede conectar otra tarjeta IMAP-VISION.

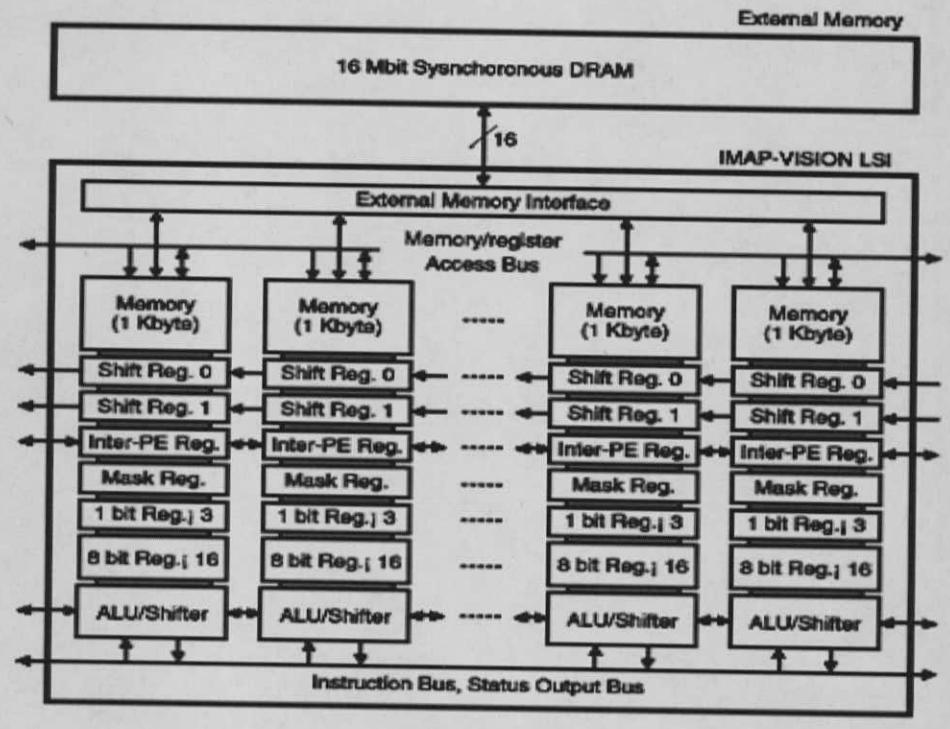


Figura 2.15: Diagrama a bloques del chip IMAP-VISION y memoria externa.

La tarjeta IMAP-VISION contiene memoria de programa, el computador anfitrión compila y ensambla el programa para posteriormente enviarlo a la memoria de programa, donde es ejecutado por el procesador de control.

Todos los elementos en la tarjeta son controlados por el procesador de control (CP), la figura 2.16 muestra el diagrama a bloques del procesador de control. Después de que el computador anfitrión escribe el programa en la memoria de programa, el procesador de control ejecuta línea por línea el programa. Cada línea consiste en una instrucción para el procesador de control y una instrucción para PE. El procesador de control ejecuta su parte y envía la instrucción a los PEs para su ejecución. El procesador de control tiene una unidad lógica aritmética de 16 bits, registro de corrimiento de 16 bits y multiplicador de 16 bits. También cuenta con 32 registros de 16-bits para propósito general, usados en el control de ciclos, cálculo de direcciones, y operaciones de variables globales. Las partes secuenciales del programa son ejecutadas por el procesador de control.

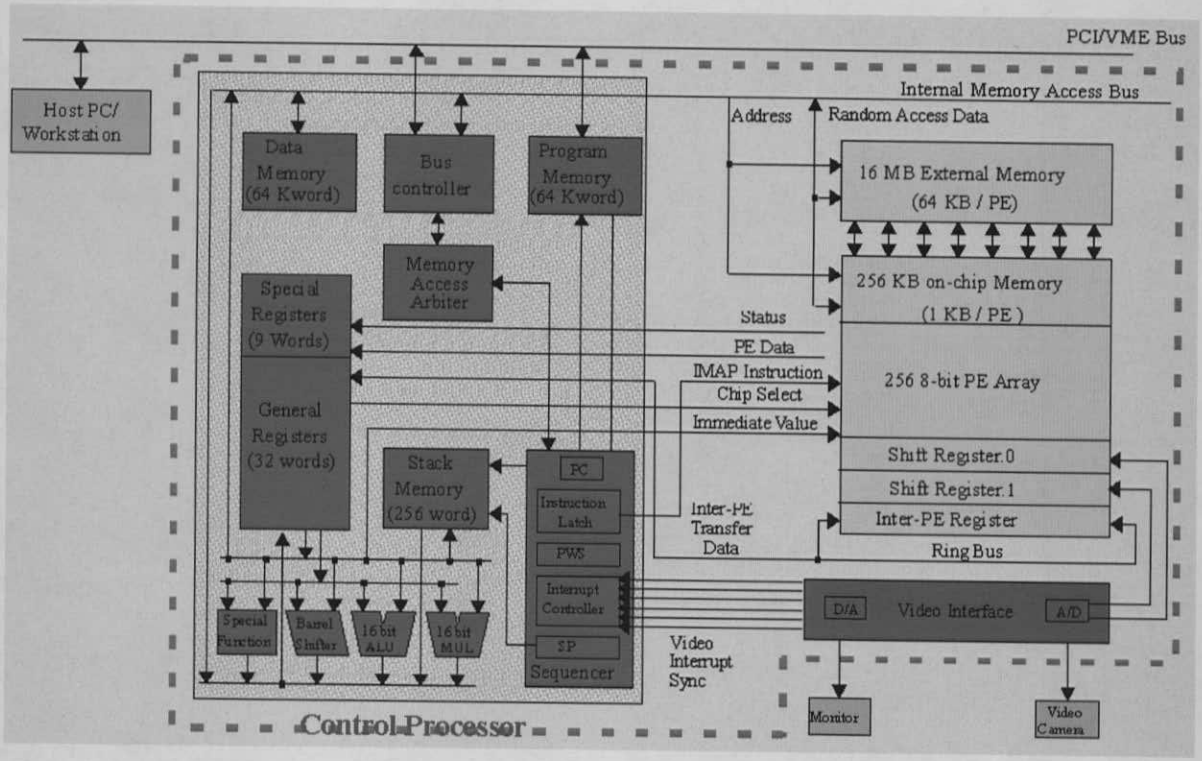


Figura 2.16: Diagrama a bloques del procesador de control.

Los diferentes elementos de la tarjeta IMAP-VISION y el computador anfitrión se conectan por medo de buses (figura 2.17). La comunicación con el computador anfitrión se realiza

por medio del bus PCI, el cual está conectado con el bus de memoria interno. El bus interno se conecta a todas las memorias y al bus externo. Un segundo bus conecta la memoria externa con la memoria interna. Este bus tiene una velocidad de transferencia de 1.28 GB/s (256x5MB/s), lo cual permite un rápido intercambio de imágenes en la memoria interna. La velocidad de transferencia entre los PEs y la memoria interna es de 10.24 GB/s (=256x40 MB/s).

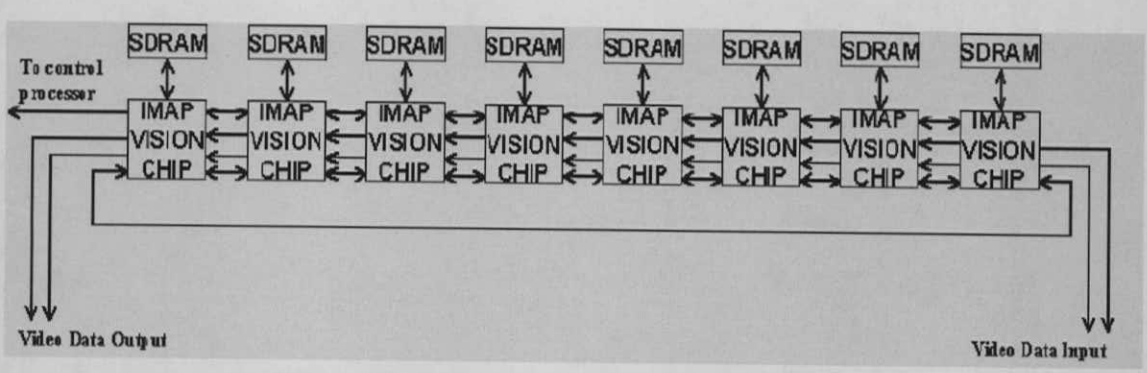


Figura 2.17: Buses en la tarjeta IMAP-VISION.

A través de cuatro buses se conectan todos los PEs. El primero conecta los PEs con el procesador de control y es usado para cargar las instrucciones y datos en los PEs. El segundo rodea el arreglo de los PEs, con este bus se conectan los PEs con su alrededor. Por medio de este bus un PE puede acceder valores del PE contiguo. El acceso a un PE lejano se puede realizar por medio de corrimientos repetitivos.

El tercero y cuarto bus conecta a los PEs al ADC y DAC de video. Para imágenes de color un bus transfiere los valores de "Y", mientras el otro transfiere los valores de Cr y Cb. A través de estos buses se trasfiere a los PEs imágenes a razón de 30 cuadros por segundo.

Las imágenes son desplazadas a través de los PEs línea por línea. Cada PE almacena el pixel de una columna. Cuando la imagen es proviene de la memoria externa cada PE almacena una columna de 240 bytes en su memoria interna, con una memoria de 1KB se puede almacenar hasta cuatro imágenes.

El lenguaje 1DC [4] desarrollado por NEC Incubation Center, es un lenguaje de programación de alto nivel para tarjetas de procesamiento paralelo independiente de la máquina. El lenguaje es basado en el lenguaje de programación ANSI "C".

El compilador 1DC (figura 2.18) de la tarjeta IMPAP-VISION se llama ccimap. Este se usa de la misma manera que se utiliza un compilador común de "C". Éste compila el archivo fuente, formando un código objeto, para después ligarlo con librerías y crear un archivo ejecutable para la tarjeta IMAP-VISION.

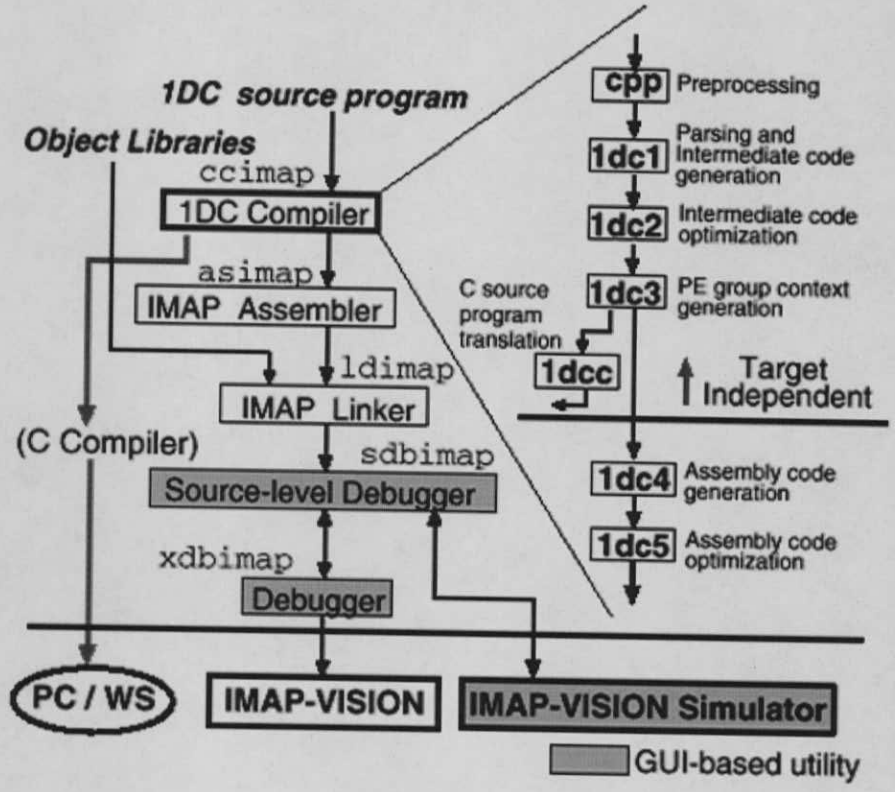


Figura 2.18: Diagrama de flujo del compilador 1DC.

La tarjeta IMAP-VISION se programa fácilmente usando el lenguaje 1DC de alto nivel, pero la programación en lenguaje ensamblador presenta mayor eficiencia. La tarjeta IMAP-VISION cuenta con su lenguaje ensamblador llamado IMAP-assembly. El ensamblador generalmente es 1.6 veces más rápido que el lenguaje 1DC.

A simple vista el lenguaje ensamblador parece complicado, esto porque cada línea lleva dos instrucciones y dos señalizadores. La primer instrucción es para el procesador de control, la segunda para el PE. Los señalizadores controlan interrupciones y configuran el registro de mascara para los PEs. Los PEs y el procesador de control son procesadores con un reducido conjunto de instrucciones (RISC), por lo que no es muy difícil la programación en lenguaje ensamblador.

2.3 Hardware de gráficos

2.3.1 OpenGL

OpenGL [27] es un software de interfaz con el hardware de gráficos. Ésta interfaz consiste de aproximadamente 150 comandos, para especificar objetos y operaciones en aplicaciones de tres dimensiones. OpenGL es una interfaz independiente del hardware la cual se puede ejecutar en diferentes plataformas.Para conseguir la independencia del hardware de OpenGL, no se incluyen comandos para tareas de ventanas, ni para obtener entrada del usuario.

Para hacer a OpenGL verdaderamente portable e independiente de la plataforma, se eliminaron todos los comandos del sistema de ventanas, por ejemplo: abrir una ventana, cerrar una ventana, escalar una ventana, dar forma a una ventana, leer la posición del cursor; y también con los dispositivos de entrada como el teclado. Todas estas acciones son altamente dependientes del sistema operativo.

Con OpenGL se pueden construir complicadas formas a partir de un conjunto de primitivas gráficas. OpenGL es una biblioteca de trazado de gráficos de alto rendimiento, varias tarjetas gráficas aceleradoras y especializadas en 3D, implementan primitivas OpenGL a nivel hardware.

Primitivas geométricas: Permiten construir descripciones matemáticas de objetos. Las actuales primitivas son: puntos, líneas, polígonos, imágenes y bitmaps.

Codificación del Color: en modos RGBA (Rojo - Verde - Azul - Alfa) o de color indexado.

Visualización y Modelado: permite disponer objetos en una escena tridimensional, mover la cámara por el espacio y seleccionar la posición deseada para visualizar la escena.
Mapeado de texturas: ayuda a dar realismo a los modelos, por medio del dibujo de superficies realistas en las caras de los modelos poligonales · La iluminación de materiales es una parte indispensable de cualquier gráfico 3D. OpenGL provee de comandos para calcular el color de cualquier punto, dadas las propiedades del material y las fuentes de luz en la habitación.

Doble buffering: ayuda a eliminar el parpadeo de las animaciones. Cada cuadro consecutivo en una animación se construye en un buffer separado de memoria, y es mostrado hasta que está completo.

Anti-alizado: reduce los bordes escalonados en las líneas dibujadas sobre una pantalla. Los bordes escalonados aparecen a menudo cuando las líneas se dibujan con baja resolución. El anti-alizado es una técnica común en gráficos de ordenador, que modifica el color y la intensidad de los pixels cercanos a la línea, para reducir el zig-zag.

Sombreado Gouraud: es una técnica usada para aplicar sombreados suaves a un objeto 3D, y producir una sutil diferencia de color por su superficie.

Z-buffering: mantiene registros de la coordenada Z de un objeto 3D. El Z-buffer se usa para registrar la proximidad de un objeto al observador, y es también crucial para el eliminado de superficies ocultas.

Efectos atmosféricos, como la niebla y el humo: hacen que las imágenes producidas sean más realistas. Sin efectos atmosféricos las imágenes aparecen a irrealmente nítidas y bien definidas. Niebla es un término que en realidad describe un algoritmo que simula neblinas, brumas, humo, polución o simplemente el efecto del aire; añadiendo profundidad a las imágenes.

Alpha blending: usa el valor Alfa (valor de material difuso) del código RGBA, y permite combinar el color del fragmento que se procesa, por el color del pixel que ya está en el buffer.

Planos de plantilla: permiten restringir el trazado a ciertas regiones de la pantalla.

Listas de Display: permiten almacenar comandos de dibujo en una lista para un trazado posterior, cuando las listas de display se usan apropiadamente pueden mejorar mucho el rendimiento de las aplicaciones.

Evaluadores Polinómicos: sirven para soportar B-splines racionales no uniformes, esto es, para ayudar a dibujar curvas suaves a través de unos cuantos puntos de referencia, ahorrando la necesidad de acumular grandes cantidades de puntos intermedios.

Características de Feedback, Selección y Elección: ayudan a crear aplicaciones que permiten al usuario seleccionar una región de la pantalla, o elegir un objeto dibujado en la misma. El modo de feedback permite al desarrollador obtener los resultados de los cálculos de trazado.

2.3.2 OpenVIDIA

El sistema de procesamiento OpenVIDIA se presenta en [8], éste se basa en múltiples unidades de procesamiento gráfico (GPU), y tiene como propósito el procesamiento de imágenes en tiempo real e implementación de algoritmos de visión por computadora. Se emplean varias tarjetas de video para operar como una arquitectura paralela de gráficos para propósito general. OpenVIDIA es construido con OpenGL y C++. OpenVIDIA cuenta con librerías para ciertas interfaces, como es la IEEE 1394 (Fire Wire), empleada para captura de video digital.

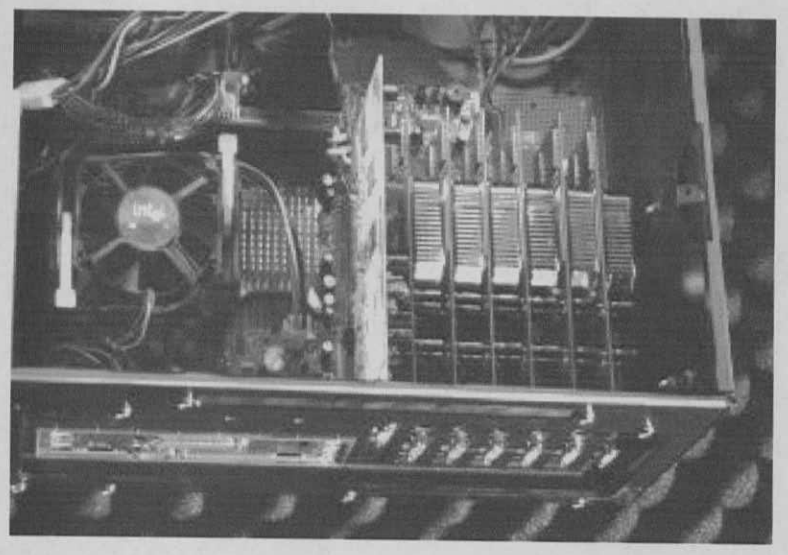


Figura 2.19: tarjeta madre con varias tarjetas de gráficos

OpenVIDIA explora en el uso del hardware de gráficos, a la inversa, para acelerar los algoritmos de visión por computador. Además OpenVIDIA explora las arquitecturas paralelas de visión por gráficos, creadas al colocar varias tarjetas de gráficos en una misma tarjeta madre. Con esto se crea fácilmente una arquitectura de bajo costo para aplicaciones

de visión por computadora y procesamiento de señales.
OpenVIDIA provee una Interfaz de Programación de Aplicaciones (API) implementada con OpenGL. La API de OpenVIDIA provee una interfaz para la creación de filtros usando el hardware de gráficos. En el paquete de OpenVIDIA están incluidos operadores para el filtrado de imágenes, detección de formas y seguimiento, visión stereo y transformada de Hough.

2.3 DSPs

Los Procesadores digitales de señales (DSPs) tienen una cualidad que los distingue de otros dispositivos de procesamiento; manejan un solo tipo de datos, "señales". Los DSPs han entrado en aplicaciones en las cuales se maneja información del mundo real. Antes de los DSPs, estas aplicaciones se implementaban a través de dispositivos analógicos.

Los DSPs realizan su función a través de la ejecución de algoritmos matemáticos, los cuales se desarrollan en cada área de aplicación. La multiplicación es la operación esencial en los DSPs, la realizan en un ciclo de reloj, ésta puede ser en punto fijo o punto flotante. Su arquitectura está desarrollada para aplicaciones en tiempo real.

Un ejemplo de procesamiento morfológico a través de DSPs se presenta en [11]. El trabajo lleva por título *"Filtrado Morfológico de Imágenes: Implementación de Primitivas Basada en DSPs"*. En este trabajo se desarrolla un algoritmo para procesamiento morfológico empleando arquitecturas paralelas a través del DSP TMS320C40 de Texas Instruments. Se exponen dos técnicas para el procesado de imágenes utilizando morfología matemática: el método clásico; actuando sobre los pixels de forma local, y la utilización de proyecciones con desplazamientos sobre la imagen completa.

En la figura 2.20 se presentan los resultados obtenidos, en esta tabla se hace una comparación entre el método de dilatación local y el método a imagen completa

Tamaño EE	*Dilatación Imagen Completa usando DSPs*	*Dilatación local usando DSPs*	*Dilatación Imagen Completa sin DSPs*	*Dilatación local sin DSPs*
Promedio 3x3	170	270	830	1040
Promedio 7x7	330	390	2200	3620
Promedio 11x11	490	770	3460	8180
Promedio 15x15	710	1090	5220	12910
Promedio 19x19	980	1320	7420	19890
Promedio 23x23	1160	1810	8180	28130
Promedio 27x27	1430	2420	9940	37460
Promedio 31x31	1640	2910	12240	47670
Promedio 35x35	1810	3630	13190	58770

Figura 2.20 tiempos (ms) de procesamiento usando DSPs

2.4 FPGAs

Los dispositivos de lógica programable más versátiles son los FPGA (*Field Programmable Gate Array*). Internamente, un FPGA esta compuesto por un conjunto de bloques lógicos iguales (CLBs), dispuestos de forma de arreglo. Cada bloque contiene pequeñas memorias RAM y flips-flops, estos componentes se pueden configurar para realizar todo tipo de circuitos combinacionales y secuenciales. Los bloques se pueden interconectar entre sí mediante conexiones también configurables. La configuración del FPGA se realiza mediante una comunicación serie denominada *bitstream*, que puede estar almacenado en una memoria externa: PROM, EEPROM, RAM; o provenir de otro sistema: PC, microcontrolador u otro FPGA. La tecnología FPGA permite realizar diseños a la medida, de bajo costo de desarrollo, incluso para la producción de pocas unidades.

En [6] se presenta un trabajo titulado "*A Design Methodology for Creating Programmable Logic-based Real-time Image Processing Hardware*". En este trabajo describe una metodología para desarrollar hardware para procesamiento de imágenes en tiempo real. La metodología se basa en dos componentes: Un hardware reconfigurable llamado *Custom Computing Machine* (CCM), éste se integra por varios FPGAs; un sofisticado Software de desarrollo denominado TRAVERSE. Éste es un sistema desarrollado específicamente para crear diseños para el procesamiento de imágenes en tiempo real basado en múltiples FPGAS (CCM).

Capítulo 3

Nociones de Morfología Matemática

Capítulo 3. Nociones de Morfología Matemática

3.1 Introducción

La Morfología Matemática se basa en la geometría y la forma, es una excelente herramienta para extraer componentes de una imagen, útiles para representar y describir la forma de una región, tales como contornos y esqueletos.

La descripción básica de la Morfología Matemática descansa en la teoría de conjuntos cuyos primeros trabajos se deben a Minkowski y Hadwiger. La continuación de estos trabajos de investigación bajo la impulsión y reformulación de Matheron y Serra, se dan posteriormente a conocer bajo la denominación de Morfología Matemática, como una técnica no lineal de tratamiento de señales.

En [16] y [17] se dedica una sección de la obra a la descripción de los operadores básicos y algoritmos más conocidos de la Morfología Matemática, abarcando el estudio blanco y negro (b&n) y tonos de gris. En [14] se dedica la obra completa a la descripción de la Morfología Matemática de Conjuntos, con un análisis completo y detallado, apoyado en abundantes demostraciones y ejemplos sencillos. En [15] la obra constituye un amplio y detallado tutorial, dedicado a imágenes en b&n, tonos de gris y casos de aplicación, la descripción se realiza matemáticamente, y a través de ejemplos presentados por medio de gráficos. En [18] se tiene un tutorial donde se encuentra la descripción de la erosión y la dilatación como una convolución booleana.

En el presente capítulo se hace una descripción de los operadores de erosión, dilatación, apertura, cerradura y contorno, tomando la información de [14] se presentan tres métodos para la obtención de la Erosión y la Dilatación: el primero a través de la suma de Minkowski, el segundo método hace uso de las traslaciones y uniones de conjuntos, el tercero permite encontrar la Dilatación a través de la reflexión del elemento estructurante como una sonda de prueba, que se traslada por todos los puntos del arreglo que contiene la imagen para verificar si se cumple o no cierta condición. El tercer método es en el que se apoya este trabajo para definir la arquitectura específica con la cual, se procesan los operadores básicos de la Morfología Matemática.

3.2 Definiciones Básicas sobre conjuntos

Definición 3.1 Dos conjuntos son iguales si y sólo si tienen los mismos elementos. Simbólicamente, si A y B son dos conjuntos, se cumple que:

$$(A = B) \leftrightarrow \forall x[(x \in A) \leftrightarrow (x \in B)]$$

Definición 3.2 La expresión $X = \{x | P(x)\}$ describe por especificación al conjunto de todos los elementos x de un conjunto U que cumplen con la función proposicional $P(x)$. El conjunto descrito X está formado por los elementos $a \in U$ para los cuales la proposición $P(a)$ es verdadera. Al conjunto U se le llama conjunto universo.

Definición 3.3 Dado un conjunto A cualquiera, se define la función característica

$XA : U \rightarrow \{0,1\}$

$$XA(a) = \begin{cases} 1 \; si \in A \\ 0 \; en \; otro \; caso \end{cases}$$

Definición 3.4 Si un conjunto A tiene exactamente k elementos diferentes, donde k es un número entero no negativo, entonces A es un conjunto finito. Se dice que la cardinalidad del conjunto A es k, y se representa así: $|A| = k$.

Definición 3.5 Un conjunto es infinito, si no es finito.

Definición 3.6 Una sucesión entera, denotada por $\{a_i\}$, es un conjunto ordenado cuyos elementos son números enteros y los índices i son números naturales ordenados, que representan el orden de aparición de los elementos; es decir, $a_i \in Z \wedge i = 0,1,2,3\ldots$

Definición 3.7 La representación gráfica de una sucesión entera recibe el nombre de histograma. En un histograma se colocan los índices en el eje convencional de las x, y los valores enteros a_i en el eje convencional de las y; después se unen con segmentos los a_i con sus proyecciones en el eje x.

Definición 3.8 El conjunto que carece de elementos se llama conjunto vacío, y el símbolo reservado para el conjunto vacío es ϕ. En ocasiones también se usa el símbolo $\{\ \}$ para

representarlo.

Definición 3.9 El conjunto A es un subconjunto del conjunto B, denotado por $A \subseteq B$, si y sólo si todo elemento de A también es elemento de B.

Simbólicamente:

$$A \subseteq B \rightarrow (x \in A \rightarrow x \in B, \forall x \in A)$$

Definición 3.10 $A = B$ si y sólo si $(A \subseteq B) \wedge (B \subseteq A)$

Definición 3.11 Si $A \subseteq B$ y $\exists x \in B \mid x \notin A$ entonces el conjunto A es subconjunto propio del conjunto B, y se denota por $A \subset B$.

Definición 3.12 Dado un conjunto X, el conjunto potencia de X es el conjunto cuyos elementos son todos los subconjuntos de X. El conjunto potencia de X se denota por 2^x.

Definición 3.13 El producto cartesiano de dos conjuntos A y B, denotado por A x B, se define como el conjunto de parejas ordenadas *(a,b)* en las que $a \in A$ y $b \in B$, es decir:

$$A \times B = \{(a,b) \mid a \in A \wedge b \in B\}$$

Definición 3.14 El producto cartesiano de n conjuntos, denotado por

$$\prod_{i=1}^{n} X_i = X_1 \times X_2 \times \ldots \times_n$$

Se define como el conjunto de la n-adas $(x_1, x_2, \ldots, x_n)$, en las que la *i-ésima* coordenada $x_i \in X_i, i = 1,2\ldots,n$, es decir:

$$\prod_{i=1}^{n} X_i = X_1 \times X_2 \times \ldots \times X_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in X_i, i = 1,2\ldots,n\}$$

Si $X = X_i, \forall i = 1,2\ldots,n$, entonces

$$\prod_{i=1}^{n} X_i = X^n$$

Definición 3.15 La unión de dos conjuntos cualesquiera X y Y, de notada por $X \cup Y$, es el conjunto que contiene a los elementos que pertenecen a X o que pertenecen a Y. Es decir:

$$X \cup Y = \{x \mid x \in X \vee x \in Y\}$$

Definición 3.16 La intersección de dos conjuntos X y Y, denotada por $X \cap Y$, es el conjunto que contiene a los elementos que pertenecen a X y que pertenecen a Y. Es decir:

$$X \cap Y = \{x \mid x \in X \wedge x \in Y\}$$

El conjunto $X \cap Y$ sólo contiene los elementos que se encuentran en X y Y de forma simultanea.

Definición 3.17 Dos conjuntos X, Y se llaman ajenos o disjuntos si y sólo si $X \cap Y = \phi$

Definición 3.18 Sean X, Y conjuntos cualesquiera. La diferencia $X - Y$ se define como el conjunto formado por aquellos elementos de X que no pertenecen a Y. Es decir:

$$X - Y = \{x \in X \mid x \notin Y\}$$

Definición 3.19 Sea X un conjunto cualquiera en un conjunto universo U. El complemento de X respecto a U (o simplemente complemento), denotado por X^c, se define como el conjunto formado por aquellos elementos de U que no pertenecen a X. Es decir:

$$X^c = \{x \in U \mid x \notin x\}$$

Definición 3.20 Sea X un conjunto cualquiera en un conjunto universo U cuyos elementos tienen inverso aditivo. La reflexión de X (o conjunto reflejado de X), cuya notación es X^-, se define como el conjunto formado por los inversos aditivos de los elementos de X: Es decir:

$$X^- = \{-x \mid x \in X\}$$

3.3 Dilatación binaria

De [14] se toman tres métodos para obtener la dilatación:

El primer método emplea la suma de Minkowski, ésta consiste en dados dos conjuntos sumar sus elementos uno a uno. El resultado final de una suma de Minkowski entre dos conjuntos X y Y, se expresa como el conjunto de los resultados de las operaciones $x + y$ donde x es elemento de uno de los conjuntos, y y es elemento del otro conjunto. La operación debe ser adecuada para el tipo de elementos que contengan los conjuntos cuyas formas se suman: si son números enteros o reales, + es la operación usual de adición; si son vectores, + es la suma vectorial, y así con cada tipo de elementos.

Definición 3.21 Sean dos conjuntos $A \subseteq X, B \subseteq X$. La dilatación de A por B, denotada por $A \oplus B$, es la suma de Minkowski de A y B; es decir, es el conjunto que resulta de sumar cada elemento de A con cada elemento de B:

$$A \oplus B = \{x = a + b \in X | a \in A \wedge b \in B\}$$

En principio, el conjunto universo X puede ser cualquier conjunto donde la operación suma esté bien definida. La dimensión del conjunto X determina la definición de suma que tiene sentido en el espacio correspondiente.

Por ejemplo, si X es el conjunto de los números enteros Z, la operación de suma es la usual entre escalares enteros; si X es el conjunto de puntos en el plano con valores enteros en sus coordenadas $Z^2 = Z \times Z = \{(x, y) | \, x \in Z \wedge y \in Z\}$, la operación de suma es la definida entre parejas ordenadas, componente a componente, o vectores del plano con coordenadas enteras.

Ejemplo 3.1 Sean $A = \{2,4,6,8,10\} \subset Z$ y $B = \{0,3\} \subset Z$. La dilatación de A por B es:

$$A \oplus B = \{x = a + b \in X | a \in A \wedge b \in B\}$$

$$A \oplus B = \{0+2, 0+4, 0+6, 0+8, 0+10, 3+2, 3+4, 3+6, 3+8, 3+10\}$$

$$A \oplus B = \{2, 4, 6, 8, 10, 11, 5, 7, 9, 11, 13\}$$

$$A \oplus B = \{2, \ 4, 5, 6, 7, 8, 9, 10, 11, 13\}$$

Comparando el conjunto A con el resultado de la dilatación, se puede ver como el conjunto A se ha dilatado, agregándose los elementos 3, 5, 7, 9, 11, 13.

Ejemplo 3.2 Sean $A = \{(0,0),(1,0),(0,-1),(1,-1),(0,1),(-1,1),(0,2),(-1,2)\}$ y $B = \{(0,0),(0,1),(0,-1)\}$ el conjunto dilatación de A por B se obtiene de la siguiente manera:

$$A \oplus B = \left\{ x = a + b \in Z^2 \mid a \in A \wedge b \in B \right\}$$

Las sumas se realizan entre parejas ordenadas de números enteros, coordenada por coordenada:

(0,0) + (0,0) = (0,0) (0,0) + (1,0) = (1,0) (0,0) + (0,-1) = (0,-1)

(0,0) + (1,-1) = (1,-1) (0,0) + (0,1) = (0,1) (0,0) + (-1,1) = (-1,1)

(0,0) + (0,2) = (0,2) (0,0) + (-1,2) = (-1,2)

(0,1) + (0,0) = (0,1) (0,1) + (1,0) = (1,1) (0,1) + (0,-1) = (0,0)

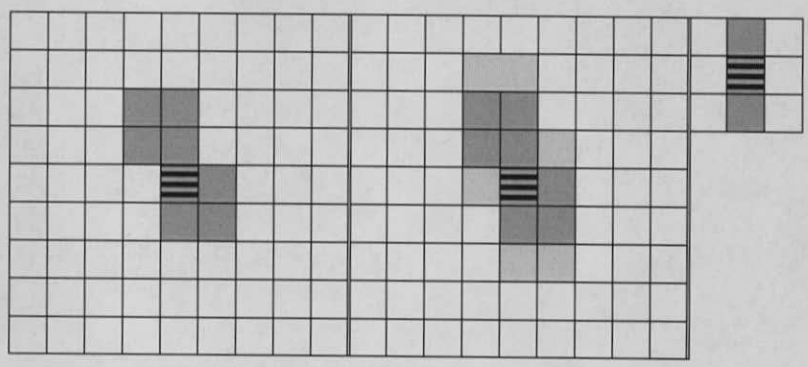
(0,1) + (1,-1) = (1,0) (0,1) + (0,1) = (0,2) (0,1) + (-1,1) = (-1,2)

(0,1) + (0,2) = (0,3) (0,1) + (-1,2) = (-1,3)

(0,-1) + (0,0) = (0,-1) (0,-1) + (1,0) = (1,-1) (0,-1) + (0,-1) = (0,-2)

(0,-1) + (1,-1) = (1,-2) (0,-1) + (0,1) = (0,0) (0,-1) + (-1,1) = (-1,0)

(0,-1) + (0,2) = (0,1) (0,-1) + (-1,2) = (-1,1)



(a) (b) (c)

Figura 3.1: Ejemplo de dilatación: (a) Imagen A, (b) Resultado de la dilatación $A \oplus B$, (c) Elemento estructurante B

En la figura 3.1 se presenta gráficamente el resultado de la Dilatación de A por B, el centro de A y B tiene una marca para distinguirlo, en (b) se muestra en tono más claro el efecto de la dilatación.

Definición 3.22 Sean: un conjunto $A \subseteq X$ y un elemento $x \in X$. La traslación de A por x se denota por $(A)_x$ y se define así:

$$(A)_x = \{y = a + x \mid a \in A\}$$

La definición 3.22 permitirá definir la dilatación como una unión de traslaciones.

Teorema 3.1 sean dos conjuntos $A \subset X$ y $B \subset X$. Se cumple que:

$$A \oplus B = \bigcup_{b \in B} (A)_b$$

Ejemplo 3.3 Hallar la dilatación de A por B aplicando el teorema 3.1.
Sean $A = \{(-1,0),(0,0),(1,0),(1,1),(0,1),(1,1),(-1,-1),(0,-1),(1,-1)\}$ y $B = \{(0,1),(-1,0),(0,0),(1,0),(0,-1)\}$

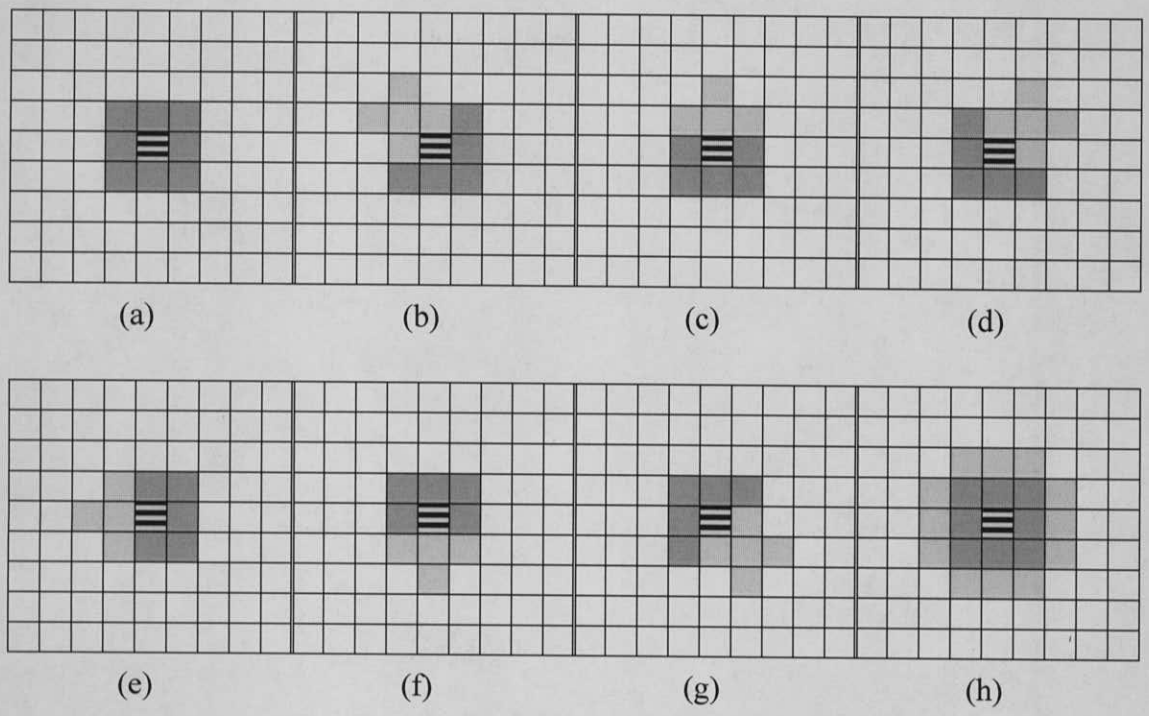


Figura 3.2 Dilatación de A por B, a través de traslaciones de B sobre A. (a) Imagen A, (b), (c), (d), (e), (f), (g) seis traslaciones de un total de nueve, (h) Dilatación completa.

La figura 3.2 muestra la imagen A, seis traslaciones y la dilatación final, observando las gráficas resulta sencillo apreciar que la unión de las traslaciones constituye el proceso de dilatación.

El tercer método hace uso de la definición 3.20 de conjunto reflejado; este método es más geométrico y permite hallar la dilatación a través del uso de la reflexión B^- del elemento estructurante como una sonda de prueba, que se traslada por todos los puntos del espacio X para verificar si se cumple o no cierta condición.

Teorema 3.2 Sean dos conjuntos $A \subset X$ *y* $B \subset X$. Se cumple que:

$$A \oplus B = \left\{ x \middle| \left(B^-\right)_x \cap A \neq \phi \right\}$$

El teorema 3.2 proporciona un método geométrico para hallar la Dilatación de un conjunto $A \subseteq X$ por un elemento estructurante $B \subseteq X$. El algoritmo consta de los siguientes pasos:

1. Escoger un punto $x \in X$

2. Hallar B^- la reflexión del elemento estructurante B

3. Encontrar $(B^-)_x$, la traslación de B^- por x

4. Realizar la operación de intersección $(B^-)_x \cap A$

5. Si el conjunto $(B^-)_x \cap A \neq \phi$ entonces $x \in A \oplus B$

6. Los pasos 1 al 5 se repiten para todos los puntos del espacio X

Este método es muy importante desde los enfoques geométrico y computacional porque permite visualizar la manera en que el conjunto reflejado del elemento estructurante viaja por el espacio de trabajo X, a la manera de una sonda imaginaria, y en cada píxel $x \in X$ se hace coincidir el origen del elemento estructurante reflejado, para luego probar si al menos uno de los píxeles de B^- coincide con uno de los píxeles de la imagen A. Si hay coincidencia, es decir $(B^-)_x \cap A \neq \phi$, entonces x forma parte de la Dilatación $A \oplus B$; en caso contrario, B^- se traslada al siguiente píxel $x \in X$.

Asignando a los puntos de la imagen representados en gris el valor uno, y a los puntos blancos que representan el fondo el valor cero, se podrá determinar la dilatación a través de la operación lógica OR. Esto es, en el método descrito recientemente paso cinco, donde se revisa si por lo menos un píxel del elemento estructurante coincide con uno de los píxeles de la imagen, la verificación se puede realizar por medio de la OR, una OR entrega una salida cero únicamente cuando todas sus entradas son cero.

3.4 Erosión

Definición 3.23 Sean dos conjuntos $A \subseteq X, B \subseteq X$ La erosión de A por B, denotada por $A \Theta B$, es la resta de Minkowsky de A por B, es decir:

$$A \Theta B = \{x \in X \mid x + b \in A, \forall b \in B\}$$

De la definición 3.23 se desprende un procedimiento para encontrar el conjunto erosión

$A\Theta B$:

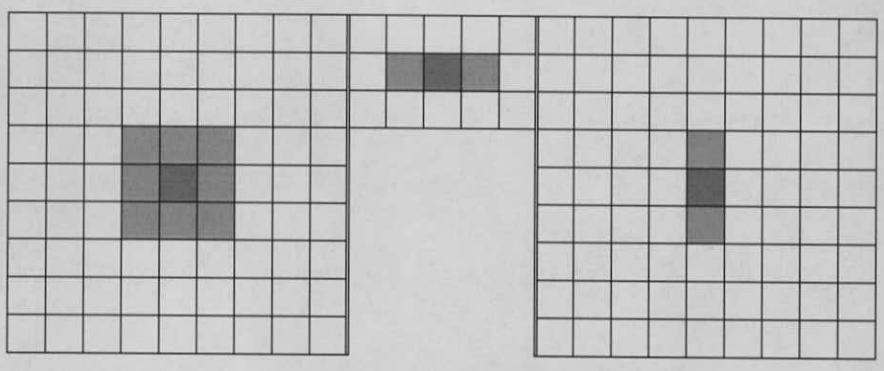
1. Escoger un punto $x \in X$

2. Sumar ese elemento x con cada uno de los elementos $b \in B$

3. Si se cumple que $x+b \in A$ para todas las sumas posibles con x fijo, entonces $x \in A\Theta B$, basta que la condición falle en una suma, para afirmar que $x \notin A\Theta B$.

4. Repetir los pasos 1, 2, 3 para todos los elementos del espacio de trabajo X

Ejemplo 3.4 Hallar la erosión $A\Theta B$ **si** $A \subseteq X \subseteq Z^2$ y $B \subseteq X \subseteq Z^2$ **son:**
$A = \{(-1,-1),(-1,0),(-1,1)(0,-1),(0,0),(0,1),(1,-1)(1,0),(1,1)\}$ **y** $B\{(-1,0),(0,0),(1,0)\}$



(a) (b) (c)

Figura 3.3 (a) imagen A, (b) elemento estructurante, (c) imagen A erosionada

El primer paso es escoger un elemento $x \in X$; se escoge el punto (4,4) que corresponde a la esquina superior izquierda, el recorrido será de izquierda a derecha y de arriba abajo.
Como segundo paso se suma el punto escogido con cada uno de los elementos de B, obteniendo tres sumas:

$$(4,4) + (-1,0) = (3,4) \notin A$$

$$(4,4) + (0,0) = (4,4) \notin A$$

$$(4,4) + (1,0) = (5,4) \notin A$$

El tercer paso consiste en verificar si el resultado de cada una de las sumas es elemento o no de A. Revisando el resultado de las sumas puede verse que ninguno de los resultados pertenece al conjunto A, por lo que se puede afirmar que

$$(4,4) \notin A \Theta B .$$

Siguiendo con el recorrido, es hasta el punto (0,1) de A donde se cumple la condición:

$$(0,1) + (-1,0) = (-1,1) \in A$$

$$(0,1) + (0,0) = (0,1) \in A$$

$$(0,1) + (1,0) = (1,1) \in A$$

Por lo anterior, la siguiente proposición es verdadera

$$(0,1) + b \in A, \forall b \in B$$

y esto significa que $(0,1) \in A \Theta B$.

En la figura 3.3 se muestran la imagen original (a), el elemento estructurante (b) y el resultado de la erosión (c).

La erosión se puede definir como una intersección de traslaciones, de manera parecida a como se definió la dilatación en términos de la unión de traslaciones. Hay una diferencia importante: las traslaciones del conjunto A ya no serán con los elementos de B, sino con los elementos del conjunto reflejado de B.

Teorema 3.3 Sean dos conjuntos $A \subset X$ y $B \subset X$. Se cumple que:

$$A \Theta B = \bigcap_{b \in B} (A)_{-b}$$

Del teorema 3.3 se desprende el siguiente método para encontrar la erosión:

1. Escoger un elemento del conjunto, digamos $b_1 \in B$

2. Encontrar $(A)_{-b_1}$, la traslación del conjunto A por el inverso aditivo del elemento $b_1 \in B$

3. Repetir los pasos 1 y 2 para cada uno de los demás elementos de B: $b_2,\ldots,b_k$

4. Encontrar el conjunto intersección de la familia $\{(A)_{-b_i}\}_{i=1,\ldots,k}$. Este conjunto es $A \Theta B$

Ejemplo 3.5 hallar la erosión del ejemplo 3.4 con el método descrito:

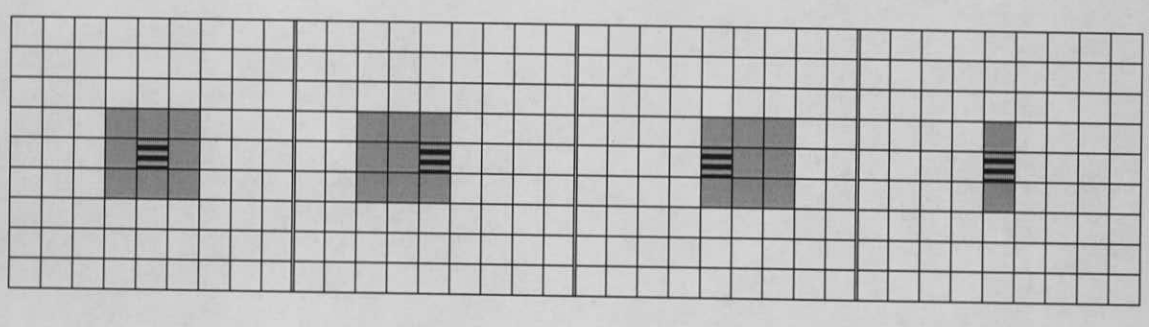


Figura 3.4 $(A)_{-(1,0)},(A)_{-(0,0)},(A)_{-(-1,0)}$ y $(A)_{-(1,0)} \cap (A)_{-(0,0)} \cap (A)_{-(-1,0)} = A \Theta B$

La figura 3.4 muestra las traslaciones y la intersección de éstas, que es la erosión $A \Theta B$.

El método de la sonda de prueba funciona también para la erosión pero, a diferencia de lo que sucede en la dilatación, no se traslada el reflejado del elemento estructurante, sino el mismo elemento estructurante B, que recorre todos los puntos del espacio X para verificar si se cumple o no cierta condición.

Teorema 3.4 sean dos conjuntos $A \subset X$ y $B \subset X$. Se cumple que:

$$A \Theta B = \{x \mid (B)_x \subseteq A\}$$

El teorema 3.4 proporciona un método geométrico para hallar la erosión de un conjunto $A \subseteq X$ por un elemento estructurante $B \subseteq X$. El algoritmo consta de los siguientes pasos:

1. Escoger un punto $x \in X$

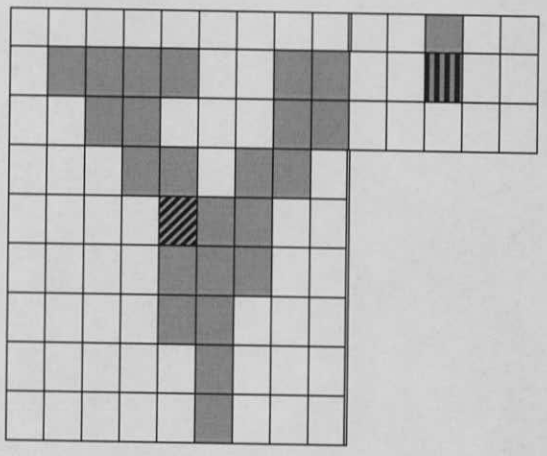
2. Encontrar $(B)_x$, la traslación de B por x

3. Si $(B)_x$ es subconjunto de A, entonces $x \in A \Theta B$

4. Repetir los pasos 1-3 para todos los puntos del espacio X

Al final se tendrá el conjunto $A \Theta B$

Ejemplo 3.6 Mediante el algoritmo anterior se determinará la erosión de la imagen A y el elemento estructurante B, representados en la figura 3.5.



(a) (b)

Figura 3.5 Imagen a erosionar(a), elemento estructurante (b).

Ahora se traslada el elemento estructurante B de manera que su origen coincida con los puntos del espacio de trabajo X de la imagen A.

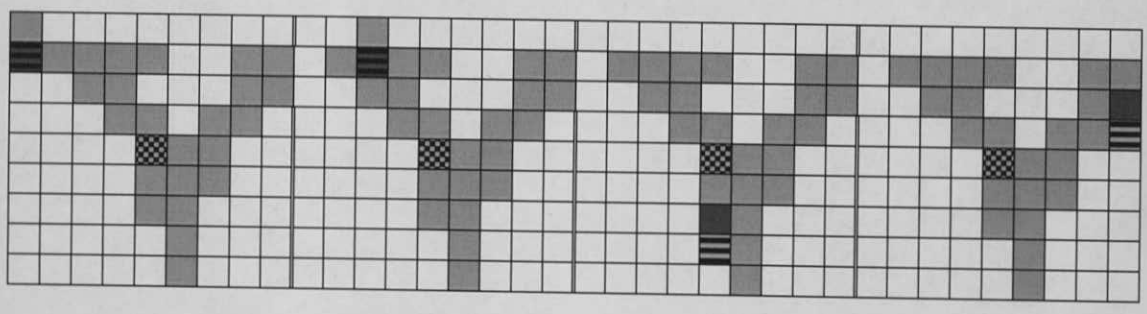


Figura 3.6 Cuatro puntos x para los que $(B)_x \not\subset A$

En la figura 3.5 se muestran traslaciones de B a cuatro diferentes $x \in X$ para los cuales se cumple que $(B)_x \not\subset A$, por lo cual estos puntos, no serán elementos de la imagen erosionada $A \Theta B$.

En la figura 3.7 (a), (b), (c) aparece la traslación de B por tres puntos que pertenecen a la erosión $A \Theta B$. En cada caso la intersección $(B)_x \cap A$ se ha graficado en tono de gris más claro. En (d) se tiene la imagen erosionada.

Representando los puntos de la imagen (grises) con uno, y los puntos del fondo (blancos) con cero, la condición $(B)_x \subseteq A$ se puede verifican utilizando la operación lógica AND, esto significa que se requiere que todos los valores de la traslación sean uno para que la salida también lo sea, indicando que $(B)_x \subseteq A$; en caso de existir un solo cero la salida es cero indicando que $(B)_x \not\subset A$.

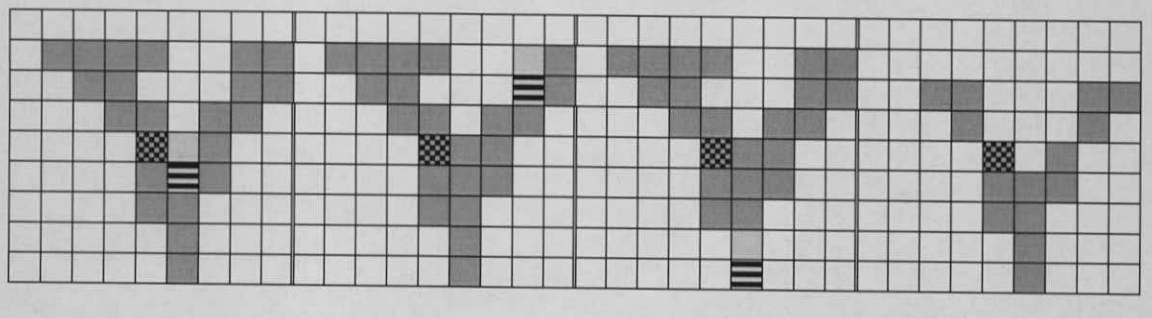
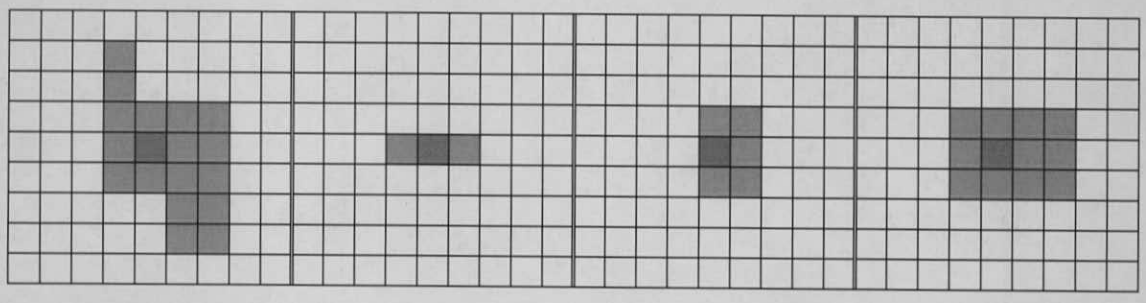


Figura 3.7 Cuatro puntos x para los que $(B)_x \subseteq A$

3.5 Apertura y cerradura

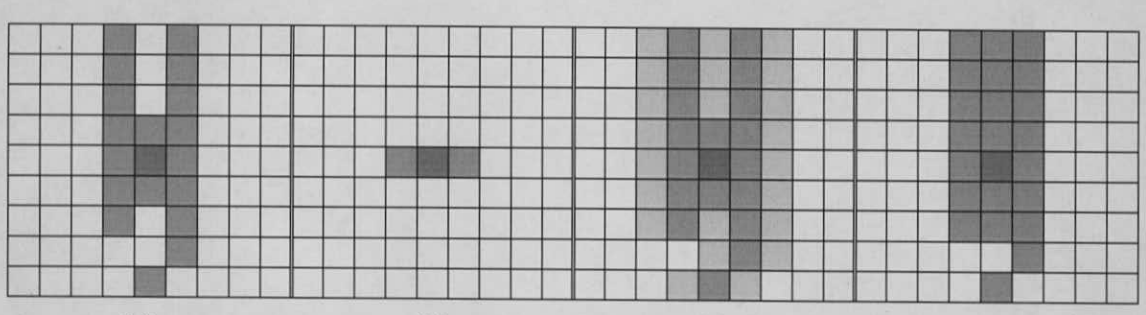
En un gran porcentaje de procesos de análisis de Conjuntos (imágenes binarias), a pesar de que la dilatación y la erosión son operaciones básicas independientes (salvo dualidad), dentro de la morfología matemática se usan por parejas alternadas; es decir, normalmente se realizan procesos iterativos del tipo: erosión seguida de una dilatación o dilatación seguida de una erosión. Se podría dudar acerca de la pertinencia de usar de esta manera la dilatación y la erosión, y la primer duda podría ser: si la dilatación está definida en términos de sumas y la erosión en términos de restas, y si la suma y la resta vectoriales son operadores duales, se podría pensar que su efecto se anula, lo anterior no ocurre y esto se ilustra a través de los dos ejemplos siguientes.



(a) (b) (c) (d)

Figura 3.8 Ejemplo de apertura. Imagen original (a), elemento estructurante (b), erosión (c), dilatación (d).

Ejemplo 3.7 La figura 3.8 presenta en (a) una imagen A, en (b) el elemento estructurante B, en (c) la imagen erosionada y en (d) la imagen dilatada con el mismo elemento estructurante. Al comparar la imagen original y la imagen $(A\Theta B)\oplus B$ se puede observar que son diferentes; esto significa que, en general la dilatación no anula los efectos de la erosión.



(a) (b) (c) (d)

Figura 3.9 Ejemplo de cerradura. Imagen original (a), elemento estructurante (b), dilatación (c), erosión (d).

Ejemplo 3.8 La figura 3.8 presenta en (a) una imagen A, en (b) el elemento estructurante B, en (c) la imagen dilatada y en (d) la imagen erosionada por el mismo elemento estructurante. Al comparar la imagen original y la imagen $(A\oplus B)\Theta B$ se puede observar que son diferentes; esto significa que, en general la erosión no anula los efectos de la dilatación.

Definición 3.24 La apertura del conjunto A por el elemento estructurante B se denota como

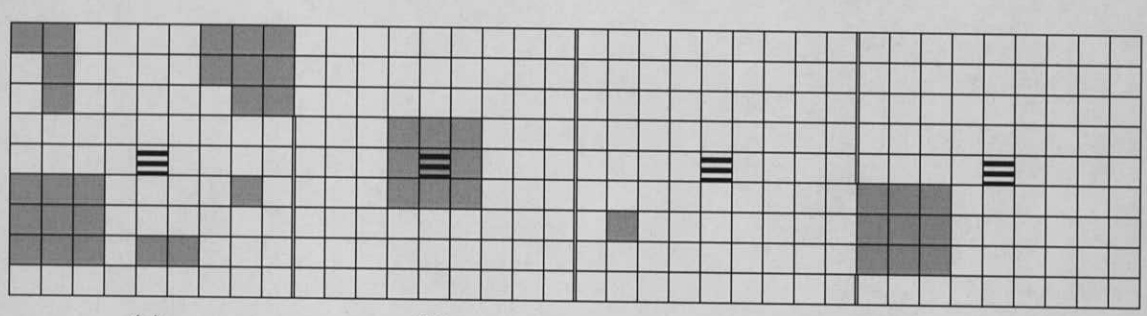
$A \circ B$ y se define así:

$$A \circ B = (A \Theta B) \oplus B$$

Los efectos de la apertura en una imagen se pueden resumir en cuatro aspectos:

1. Elimina islas de tamaño menor al elemento estructurante.
2. Elimina picos o cabos más delgados que el elemento estructurante.
3. Rompe istmos cuya anchura sea menor al diámetro del elemento estructurante.
4. Alisa el contorno convexo de la imagen.

Ejemplo 3.8 La figura 3.10 muestra un imagen A con islas. Al realizar la apertura de A con el elemento estructurante B, se puede observar que se eliminan aquellas islas con tamaño menor al elemento estructurante B. En (a) se encuentra la imagen original, en (b) el elemento estructurante, en (c) la erosión $A \Theta B$ y en (d) la apertura $A \circ B = (A \Theta B) \oplus B$.



(a) (b) (c) (d)

Figura 3.10 Apertura. (a) imagen original, (b) elemento estructurante B, (c) Erosión $A \Theta B$, (d) apertura de $A \circ B = (A \Theta B) \oplus B$ A por B

Definición 3.25 La cerradura del conjunto A por el elemento estructurante B se denota por $A \bullet B$ y se define así:

$$A \bullet B = (A \oplus B) \Theta B$$

La cerradura produce los siguientes efectos:

1. Rellena lagos o huecos de tamaño menor al elemento estructurante.

2. Rellena rajaduras o golfos más delgados que el elemento estructurante.

3. Funde estrechos cuya anchura sea menor al diámetro del elemento estructurante.

4. Alisa el contorno Cóncavo de la imagen, rellenando rompimientos.

Ejemplo 3.9 La figura 3.11 Muestra una imagen con golfos lagos y un estrecho. Al realizar la cerradura de A con B, se observa que se rellenan los lagos y golfos de tamaño menor al elemento estructurante; además el estrecho se funde y se unen las dos partes de la imagen que originalmente se encontraban separadas.

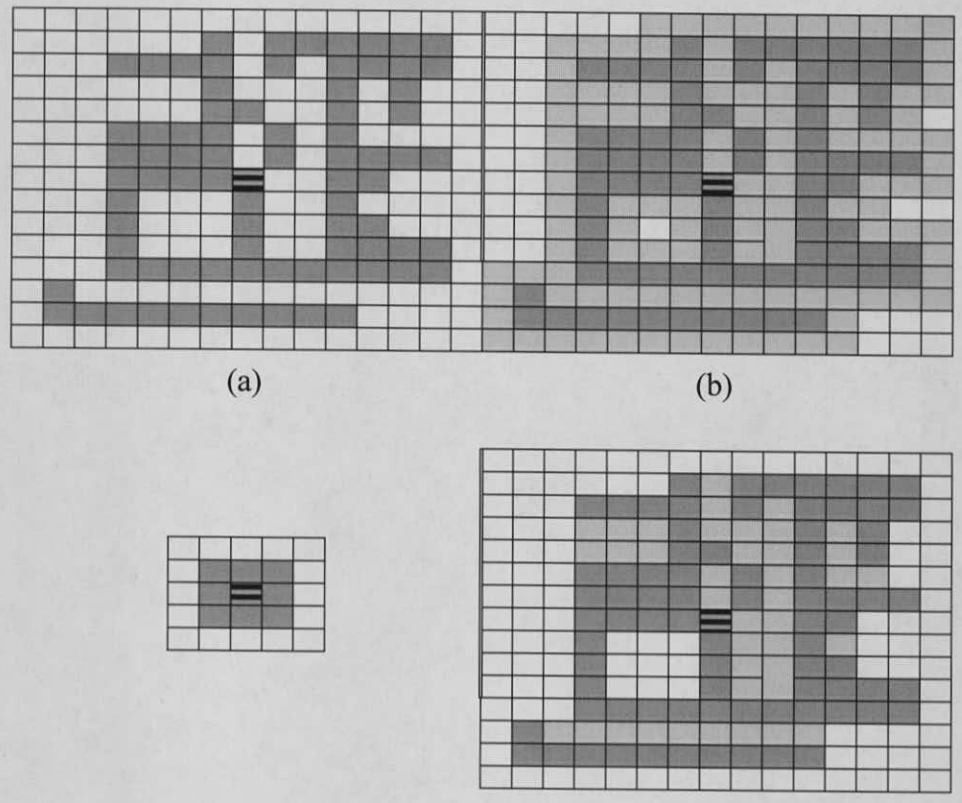


Figura 3.11 Cerradura: Imagen original (a), dilatación, elemento estructurante (c), cerradura de A por B (d).

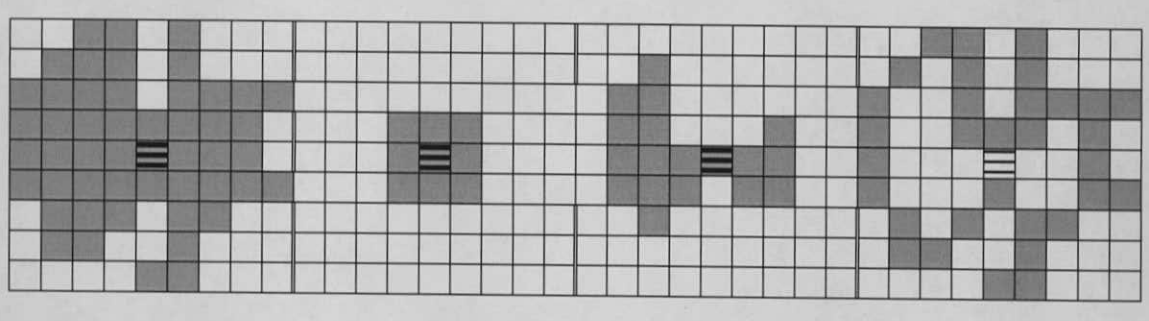
3.5.1 Idempotencia y dualidad

Similarmente a como ocurre entre la erosión y la dilatación, es posible construir la cerradura por medio de la apertura y viceversa a través de una propiedad importante que es la dualidad entre operadores. Adicionalmente, estos operadores comparten una propiedad interesante: la *idempotencia.* Esta propiedad significa que una vez realizada la apertura o la cerradura, la repetición de dicho proceso no tiene ningún efecto sobre el conjunto de trabajo. En otras palabras, no tiene sentido efectuar aperturas o cerraduras repetidas con un mismo elemento estructurante, pues resulta idéntico a efectuar una única apertura o una única cerradura.

3.6 Extracción de contornos

Teorema 3.5 Dados A, B subconjuntos de X, la frontera interna de A dado B es equivalente a $\Gamma(A) = A - (A \Theta B)$

La figura 3.12 ilustra la mecánica de la extracción de contornos: en (a) se tiene la imagen original, en (b) el elemento estructurante, en (c) la erosión de A por B, en (d) el contorno $\Gamma(A) = A - (A \Theta B)$



(a) (b) (c) (d)

Figura 3.12 Extracción contorno: (a) imagen original, (b) elemento estructurante, (c) erosión de A por B, (d) contorno de A

Considerando valores de uno para los puntos de la imagen y cero para el fondo la operación del contorno se puede evaluar a través de operadores lógicos esto es, para $-(A \Theta B)$ se utiliza la AND y un inversor para lograr el complemento, para $\Gamma(A) = A - (A \Theta B)$ se aplica la XOR entre A y la salida del inversor.

Capítulo 4

Puerto de video digital y FPGAs

Capítulo 4: Puerto de video digital y FPGAs

4.1 Introducción

Los sistemas de televisión han sido los pioneros en el manejo de señales de video, en nuestro país se utiliza el sistema NTSC (*National Television System Committee).* NTSC es un sistema de codificación y transmisión de televisión analógica, desarrollado en Estados Unidos en 1940. El sistema NTSC maneja el video en modo *entrelazado,* en éste se maneja un total de 525 líneas, 262 y 263 por campo par e impar, la velocidad de actualización es de 30 cuadros por segundo (60 campos entrelazados).

En este trabajo se aplica procesamiento en forma local (el punto y su vecindad), sin almacenar cuadros, esta situación pone de manifiesto que una señal entrelazada no es útil a nuestro propósito; para nuestro trabajo se requiere una señal de video progresiva.

Los formatos de video empleados en computadoras (a partir del VGA) manejan imágenes progresivas de mayor resolución en comparación con los sistemas de televisión, motivo por el cual en este trabajo de Tesis el procesamiento se aplica a señales de video en formato VGA.

Los dispositivos de despliegue (CRTs) usados en computadoras son analógicos, éstos requieren una señal de video analógica, en la tarjeta de video de la computadora se realiza el proceso de conversión digital - analógico. El puerto de la tarjeta de video entrega una señal analógica, por lo tanto para procesar digitalmente esta señal se requiere la conversión analógica – digital.

Recientemente se han insertado en el mercado dispositivos digitales de despliegue llamados LCDs, los cuales han mantenido por compatibilidad la entrada de video en formato analógico, internamente se realiza la conversión analógica – digital. Por su parte las tarjetas de video han evolucionado en su desempeño, y ahora es posible encontrar tarjetas de video con salida analógica y digital. La salida digital es la asignada para conectarse con LCDs, ésta recibe el nombre de DVI (Digital Visual Interface) [19], por este puerto se transmiten los valores del rojo, verde, azul (RGB), sincronía vertical y horizontal en forma serial. En la

sección 4.2 se hace una descripción de este puerto y del circuito integrado encargado del proceso de decodificación de la señal (conversión serie - paralela).

La arquitectura de procesamiento propuesta en esta Tesis se puede implementar a través de un circuito ASIC (*Application-Specific Integrated Circuit),* desarrollar este tipo de circuitos es una solución a la medida, dado que se puede integrar el hardware digital y analógico necesario para implantar en un solo chip todas las funciones requeridas. El desarrollo de un chip ASIC requiere de habilidades, herramientas CAD, y los servicios de una fundidora para plasmar el diseño en silicio. Los costos de las licencias del software y los servicios de la fundidora hacen prácticamente imposible su realización.

Los FPGAs son dispositivos de configuración flexible, con un gran número de compuertas lógicas y frecuencias de operación alrededor de los 500 Mhz, su costo es bajo y algunos fabricantes proporcionan versiones gratuitas de su software de desarrollo. Lo anterior convierte a los FPGAS en la alternativa viable para plasmar chips a la medida. Actel, Xilinx y Altera son los fabricantes de FPGAs más importantes, Xilinx ha dotado a sus FPGAs de las familias [20] Spartan y Virtex con abundantes registros y la posibilidad de usar las LUT (Look-Up Table) [24] como registros de corrimiento, lo que permite implementar éste tipo de registros aprovechando al cien por ciento los recursos del FPGA. En la sección 4.3 se describe detalladamente el FPGA Spartan II usado en este trabajo.

4.2 Puerto de video digital

En esta sección se describen las principales características del formato de video DVI, la información se obtiene de [19], éste es el documento generado por The Digital Display Working Group Promoters (DDWG Promoters), el grupo lo integran: Intel Corporation, Silicon Image Inc., Compaq Computer Corporation, Fujitsu Limited, Hewlett-Packard Company, International Business Machines Corporation, y NEC Corporation.

El puerto de video digital **DVI** es un formato de video diseñado para obtener la máxima calidad de visualización posible en pantallas digitales tales como: monitores de cristal líquido y proyectores digitales.

El puerto DVI se basa en el formato serial PanelLink, desarrollado por Silicon Image Inc, emplea TMDS (Transition Minimized Differential Signaling). Un enlace DVI consiste en un cable con cuatro pares trenzados, uno para cada color primario (rojo, verde y azul) y otro

para la señal de reloj. La sincronización de la señal es similar a la transmisión analógica de vídeo. La imagen se transmite línea por línea con intervalos de borrado entre cada línea y entre cada cuadro. No se usa compresión ni transmisión por paquetes, la pantalla entera se transmite constantemente.

El conector DVI admite un segundo enlace, con otro conjunto de pares trenzados para el rojo, el verde y el azul. Cuando se requiere un ancho de banda mayor que el que permite un solo enlace, el segundo se activa, y los dos pueden emitir pixeles alternos. El estándar DVI especifica un límite máximo de 165 MHz. para los enlaces únicos, de forma que los modos de pantalla que requieran una frecuencia inferior deben usar el modo de enlace único, y los que requieran más deben establecer el modo de enlace doble. El segundo enlace también se puede usar cuando se necesiten más de 24 bits por píxel, en cuyo caso transmite los bits menos significativos.

Al igual que los conectores analógicos VGA modernos, el conector DVI tiene pines para el canal de datos de pantalla, versión 2 (DDC 2) [25] que permite al adaptador gráfico leer los datos de identificación de pantalla extendidos (EDID, Extended Display Identification Data). Estos datos se encuentran grabados en una memoria ROM serial la cual se incluye en display.

4.2.1 Conector DVI

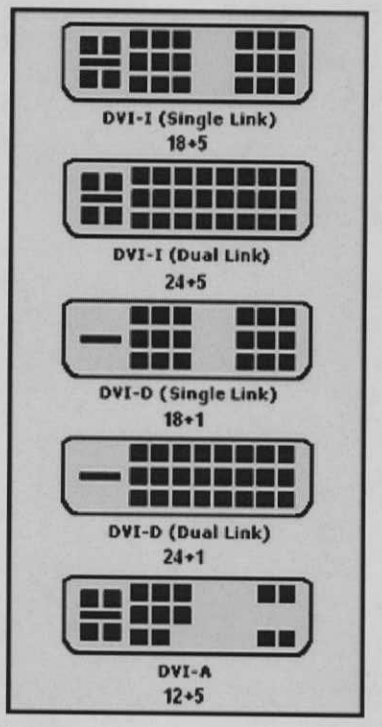


Figura 4.1: Conector DVI (conector macho)

En los sistemas de doble enlace se proporcionan pines adicionales para la segunda señal, también se tienen pines para transmitir las señales analógicas del estándar VGA. Los conectores (figura 4.1) se clasifican en tres tipos en función de qué señales admiten: **DVI-D** (digital), **DVI-A** (analógico), **DVI-I** (digital y analógico)

4.2.2 Pines del conector

En la figura 4.2 se tiene la distribución de los pines en un conector hembra DVI, se acostumbra utilizar conectores Macho-Macho en los cables DVI, y conectores hembra en las tarjetas de video, Flat Panel, y demás dispositivos que tengan un puerto DVI.

Números de pin (vista del enchufe hembra)										
1	2	3	4	5	6	7	8		C1	C2
9	10	11	12	13	14	15	16		C5	
17	18	19	20	21	22	23	24		C3	C4

Figura 4.2: Distribución de pines en el conector hembra

En la tabla 4.1 se tiene la descripción de las señales para cada uno de los pines que integran el conector DVI.

Tabla 4.1: Señales en conector DVI

Funciones de los pines		
Pin	**Nombre**	**Función**
1	Datos TMDS 2-	Rojo digital - (Link 1)
2	Datos TMDS 2+	Rojo digital + (Link 1)
3	Protección datos TMDS 2/4	
4	Datos TMDS 4–	Verde digital – (Enlace 2)
5	Datos TMDS 4+	Verde digital + (Enlace 2)
6	Reloj DDC	
7	Datos DDC	
8	Sincronización vertical analógica	
9	Datos TMDS 1–	Verde digital – (Enlace 1)
10	Datos TMDS 1+	Verde digital + (Enlace 1)
11	Protección datos TMDS 1/3	
12	Datos TMDS 3–	Azul digital – (Enlace 2)
13	Datos TMDS 3+	Azul digital + (Enlace 2)
14	+5V	Energía para el monitor en espera
15	Tierra	Retorno para pin 14 y sincronización analógica
16	Detección Hot Plug	
17	Datos TMDS 0–	Azul digital – (Enlace 1) y sincronización digital
18	Datos TMDS 0+	Azul digital + (Enlace 1) y sincronización digital
19	Protección datos TMDS 0/5	
20	Datos TMDS 5–	Rojo digital – (Enlace 2)
21	Datos TMDS 5+	Rojo digital + (Enlace 2)
22	Protección reloj TMDS	
23	Reloj TMDS+	Reloj digital + (Enlaces 1 y 2)
24	Reloj TMDS–	Reloj digital – (Enlaces 1 y 2)
C1	Rojo analógico	
C2	Verde analógico	
C3	Azul analógico	
C4	Sincronización horizontal analógica	
C5	Tierra (analógico)	Retorno para señales de Rojo, Verde y Azul

4.2.3 Decodificación del formato DVI

Para la conversión de DVI a paralelo se utilizan circuitos decodificadores como el SIL161B, SIL163B, de Silicon Image, o el TFP101, TFP401 de Texas Instruments, en [21], [22] y [23] se tienen las hojas de datos de estos dispositivos. Estos chips reciben la señal de RGB y sincronía, en transmisión TDMS para convertirla a RGB paralela, en uno o dos píxeles por pulso de reloj. La funcionalidad del circuito se puede contemplar de manera global a través del diagrama a bloques del TFP401 (figura 4.3).

A la izquierda se tienen las señales TMDS por pares en modo diferencial, el par RX0 recibe los bits correspondientes al azul y las señales de sincronía horizontal (HSYNC) y vertical (VSYNC). EL par RX1 recibe los bits del verde y la señal CTL1. El par RX2 recibe los bits correspondientes al rojo y las señales CTL2 y CTL3. El par RXC recibe la señal de reloj de referencia.

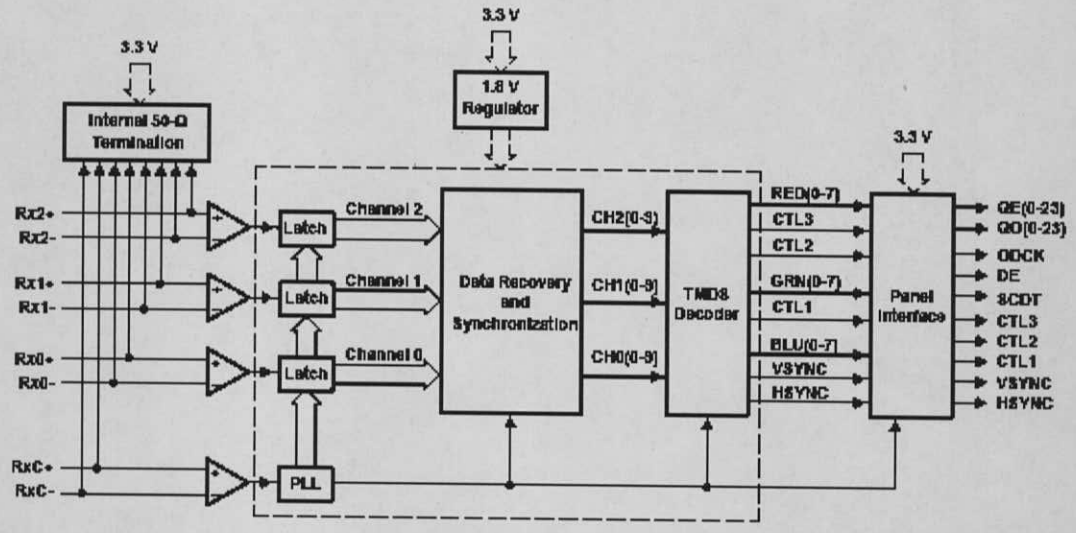


Figura 4.3: Diagrama a bloques decodificador DVI

A la derecha se tienen las señales de salida, QE[23:0] contiene los 24 bits del RGB correspondientes, 8 por color, esto en enlace sencillo o doble, QO[23:0] contiene los 24 bits del RGB en el modo de doble enlace. La señal Data Enable (DE) es activa cuando existen datos disponibles, ODCK corresponde pulso de reloj por píxel.

4.3 Características principales de la familia de FGAs Spartan-II de Xilinix.

La familia de FPGAs (Field-Programmable Gate Array) Spartan-II brinda versatilidad y abundantes recursos lógicos. Ofrece densidades de 15,000 a 200,000 compuertas, como se muestra en la tabla 4.2 El sistema soporta frecuencias de 200 MHz. Incluye bloques de memoria RAM de hasta 56k bits, RAM distribuida de hasta 75,264 bits, 16 bloques de entrada-salida estándar, y cuatro DLL.

Tabla 4.1 Características de FPGAs Spartan II

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O(1)	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	92	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	176	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

La familia de FPGAs Spartan II tiene una arquitectura (figura 4.4) flexible y programable de bloques lógicos configurables (CLBs), se encuentra rodeado de bloques de entrada - salida (IOBs) programables, cuenta con cuatro Delay-Locked Loops (DLLs), uno en cada esquina. Dos bloques de RAM en forma de columna, ubicados en lados opuestos del dispositivo, entre los CLBs y las columnas de IOB. Estos elementos son interconectados por canales de ruteo.

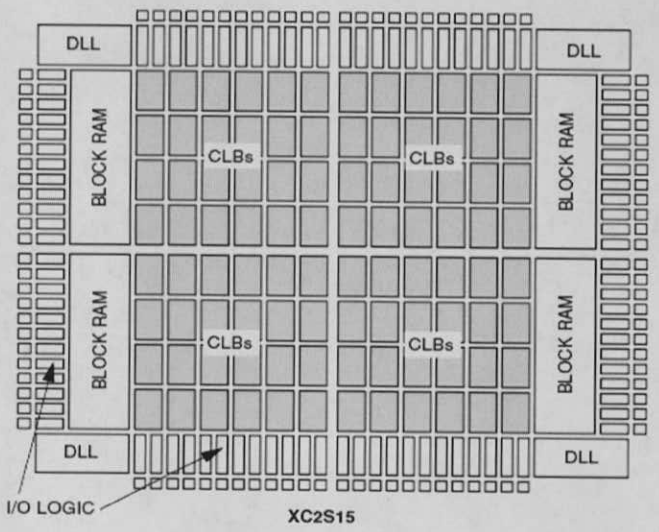


Figura 4.4: Estructura general

Los FPGAs Spartan-II son arreglados cargando los datos de configuración en celdas internas de memoria estática. Se pueden reprogramar un número ilimitado de veces. Los datos almacenados en las celdas de memoria determinan las funciones lógicas e implementan las conexiones en el FPGA. Los datos de configuración pueden ser leídos de un PROM serial externo (master serial mode), o con Boundary Scan modes, con el FPGA en modo esclavo serial (slave serial) o modo esclavo paralelo (slave parallel).

Los FPGAs Spartan-II ofrecen bloques de RAM de puerto sencillo y doble, manejadores de reloj para los DLL, todos los flip-flops cuentan con SET y RESET programable, señal de carry de rápida propagación.

4.3.1 Bloques de entrada/salida

Los IOB del Spartan-II mostrados en la figura 4.5, soportan una amplia variedad de señales estándar. Sus entradas y salidas de alta velocidad le permite soportar varios tipos de buses y memorias. La tabla 4.2 muestra los posibles tipos de señales que soporta. Los tres registros del IOB funcionan como flip-flops tipo D, sensibles al flanco o como latches sensibles al nivel. Cada IOB tiene su entrada de reloj común a los tres registros, y su entrada de clock enable independiente en cada uno. Adicionalmente a las señales de CLK y CE, los tres registros comparten la señal de SET RESET (SR). Para cada registro, esta señal puede configurarse de manera independiente como set y reset síncrono o como preset o clear asíncrono. La característica no mostrada en el diagrama de la figura 4.5, pero controlada

por software, es el control de polaridad. La entrada y salida de los buffers y todas las señales de los IOB tienen control independiente de polaridad.

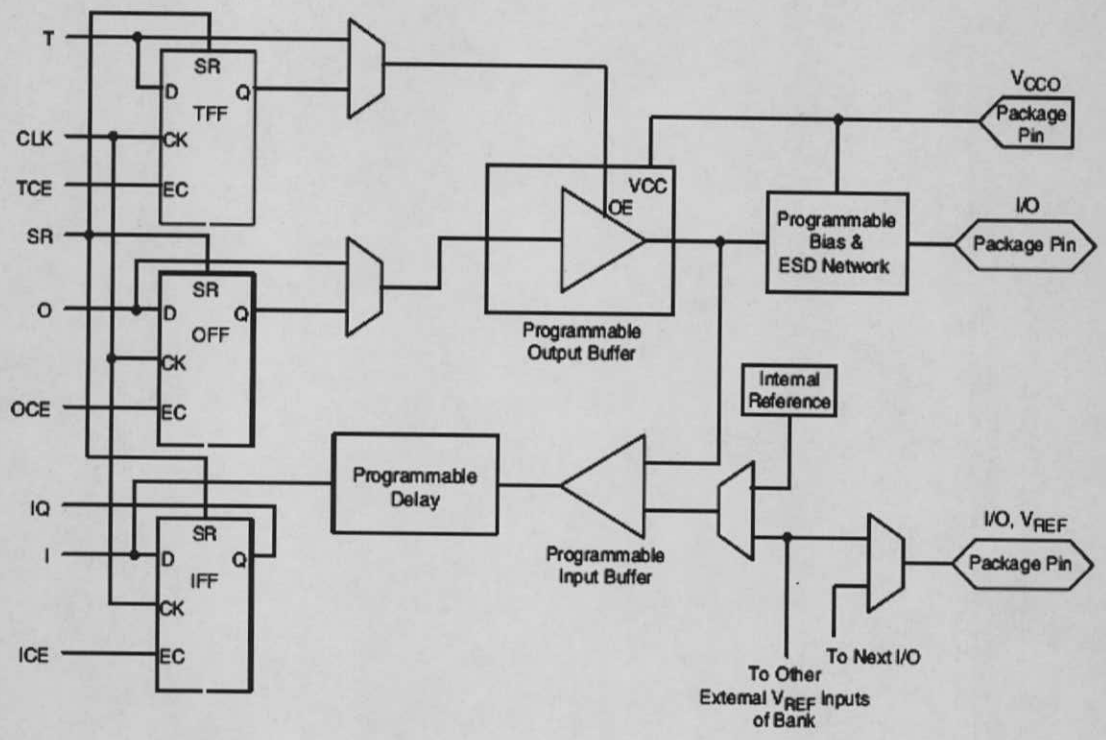


Figura 4.5: Diagrama de IOB

Tabla 4.2: Tipos de señales de entrada salida soportados

I/O Standard	Input Reference Voltage (V_{REF})	Output Source Voltage (V_{CCO})	Board Termination Voltage (V_{TT})
LVTTL (2-24 mA)	N/A	3.3	N/A
LVCMOS2	N/A	2.5	N/A
PCI (3V/5V, 33 MHz/66 MHz)	N/A	3.3	N/A
GTL	0.8	N/A	1.2
GTL+	1.0	N/A	1.5
HSTL Class I	0.75	1.5	0.75
HSTL Class III	0.9	1.5	1.5
HSTL Class IV	0.9	1.5	1.5
SSTL3 Class I and II	1.5	3.3	1.5
SSTL2 Class I and II	1.25	2.5	1.25
CTT	1.5	3.3	1.5
AGP-2X	1.32	3.3	N/A

4.3.3 Look-Up Tables

Las funciones generadas en Spartan-II son implementadas como Look-up tables (LUTs) [24] de cuatro entradas. Adicionalmente las LUT pueden proveer una RAM síncrona de 16 x 1 bit, las dos LUT de una slice pueden combinarse para formar RAM síncrona de 16 x 2 o 32 x 1 bit o RAM de doble puerto de 16 x 1.

Con las LUT del Spartan-II también se pueden formar registros de corrimiento de 16 bits, esto es ideal para la captura de datos en ráfaga. Este modo también se puede usar para almacenar datos en aplicaciones tales como procesamiento digital de señales digitales.

La primitiva SRL16 corresponde al uso de las Look-up tables como registro de corrimiento de 16 bits. El uso de este modo de registro proporciona versatilidad y optimización de recursos, reduciendo el tamaño del dispositivo y con ello el costo de implementación. La generación de Spartan II puede configurar las look-up table (LUT) en registros de corrimiento de 16 bits sin hacer uso de los flip-flops disponibles en cada bloque lógico. Las operaciones de corrimiento se realizan en forma síncrona, la longitud del registro se puede modificar desde un bit hasta 16 bits, esto a través de 4 líneas de selección, cuenta con una salida dedicada para realizar la cascada entre registros y poder lograr registros de cualquier tamaño.

4.3.4 Librería de Primitivas

Ocho primitivas están disponibles en la librería, éstas ofrecen un opcional clock enable (CE), reloj invertido (/CLK), y salidas casqueables (Q15). La tabla 4.3 lista todas las primitivas disponibles para síntesis y simulación.

Tabla 4.3 Primitivas de registros de corrimiento disponibles en librería

Primitive	Length	Control	Address Inputs	Output
SRL16	16 bits	CLK	A3, A2, A1, A0	Q
SRL16E	16 bits	CLK, CE	A3, A2, A1, A0	Q
SRL16_1	16 bits	CLK	A3, A2, A1, A0	Q
SRL16E_1	16 bits	$\overline{\text{CLK}}$, CE	A3, A2, A1, A0	Q
SRLC16	16 bits	CLK	A3, A2, A1, A0	Q, Q15
SRLC16E	16 bits	CLK, CE	A3, A2, A1, A0	Q, Q15
SRLC16_1	16 bits	CLK	A3, A2, A1, A0	Q, Q15
SRLC16E_1	16 bits	$\overline{\text{CLK}}$, CE	A3, A2, A1, A0	Q, Q15

Señales en el registro de corrimiento

La figura 4.6 muestra el diagrama a bloque del registro de corrimiento, sus señales se describen a continuación:

Reloj — CLK

El flanco de subida o bajada es usado para el corrimiento síncrono de datos.

Dato de entrada — D

El dato de entrada provee el nuevo bit a desplazar por el registro de corrimiento.

Clock Enable — CE (opcional)

La habilitación de reloj afecta la funcionalidad del corrimiento. Cuando esta señal está inactiva no hay corrimiento de datos, y no se escribe nuevo dato. La activación de esta señal permite la escritura de nuevo dato en la terminal de entrada D y el corrimiento en todo el registro.

Líneas de dirección — A3, A2, A1, A0

Las líneas de dirección permiten seleccionar un rango de 0 a 15, con estas líneas se puede variar la longitud del registro de 1 a 16 bits, cuando todas las líneas tienen cero la longitud seleccionada es uno. Las líneas de selección no tienen efecto en la salida Q15, usada para realizar la cascada entre registros.

Salida de datos — Q

La salida de datos Q entrega los datos correspondientes al registro cuya longitud se seleccionó con las líneas de dirección.

Salida de datos — Q15 (opcional)

Esta salida siempre entrega el último bit correspondiente a Q15, el cambio en la longitud del registro no afecta esta salida.

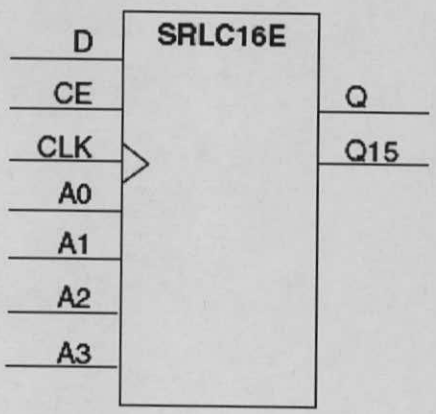


Figura 4.6: Señales en registro de corrimiento

Atributos

Valor inicial — INIT

El atributo INIT define el contenido inicial del registro de corrimiento, éste se codifica en hexadecimal con cuatro dígitos (0000). EL dígito de la izquierda corresponde al bit más significativo, por defecto el registro de corrimiento es inicializado con cero, pero durante la configuración del dispositivo otro valor puede ser especificado.

Capítulo 5

Arquitectura propuesta e Implementación

Capítulo 5: Arquitectura propuesta e Implementación

5.1 Arquitectura propuesta

Como se ha descrito en el capítulo 3 las operaciones morfológicas básicas (Erosión y Dilatación) se pueden realizar a nivel local, empleando un pixel y su vecindad, para una vecindad de un punto, se requieren nueve pixeles (3 x 3) ordenados en tres renglones por tres columnas, para definir el elemento estructurante deseado. De esta manera se puede iniciar el procesamiento sin requerir la imagen completa.

En la transmisión de señales de video la información se transmite en forma serial, esto quiere decir que la imagen llega pixel a pixel. La imagen se forma de izquierda a derecha y de arriba abajo. Esta situación nos permite ver que a partir de tres líneas de la imagen que se reciban se puede iniciar el procesamiento de la misma, de esta manera la imagen se puede procesar en forma de señal sin tener que capturarla.

La figura 5.1 muestra la arquitectura propuesta para las operaciones de erosión y dilatación, con un elemento estructurante de 3 x 3. Esta arquitectura se integra por dos registros de corrimiento, con una longitud igual a la línea de vídeo (en pixeles) y un tercero de longitud igual a tres pixeles, dado que de la tercera es suficiente contar con sus tres primeros pixeles.

Con los primeros tres píxeles de las tres líneas se forma el área de procesamiento, a partir de aquí se genera el primer punto procesado. Para la obtención de la dilatación se aplica la OR al punto y su vecindad, en el caso de la erosión se aplica la AND. La salida de las compuertas corresponde a la señal procesada. Para procesar el siguiente píxel, se realiza un corrimiento en los registros, y así hasta desplazar toda la imagen a través de los registros.

Como puede apreciarse, el tiempo de procesamiento, es igual al tiempo de propagación de la compuerta lógica más el retrazo correspondiente a la duración de dos líneas de video y tres puntos de una tercera. Después de llenar los registros el tiempo de procesamiento es prácticamente cero.

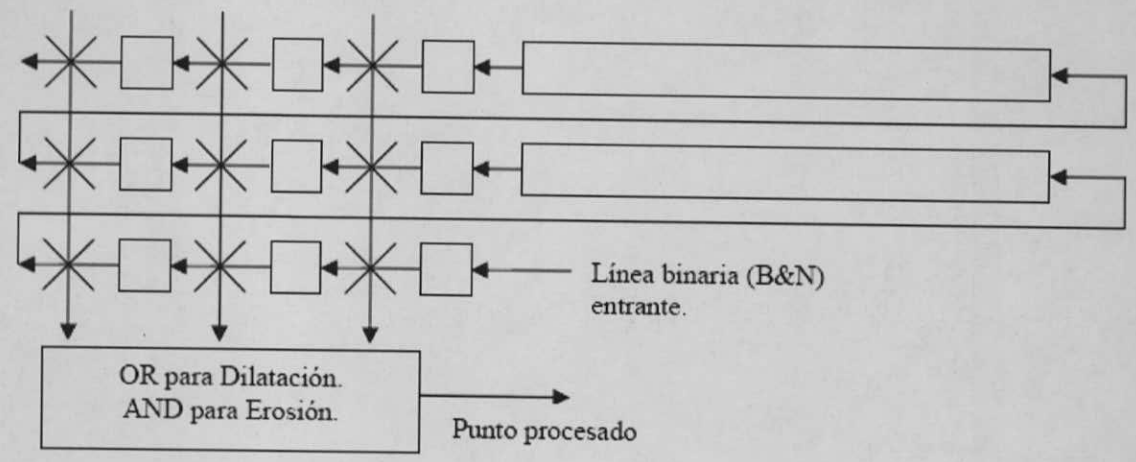


Figura 5.1: Arquitectura propuesta

A simple vista la idea es sencillamente funcional, lo que no es muy obvio es la manera de implementar esta arquitectura con dispositivos electrónicos comerciales, dado que el costo del diseño de un circuito específico es muy alto para este propósito.

Es evidente que el dispositivo mayormente requerido para este trabajo es el registro de corrimiento, un dispositivo fácil de conseguir en el mercado de componentes electrónicos, la cantidad requerida de estos depende de la longitud de línea o tamaño horizontal de la imagen, se tiene el propósito de manejar formatos de alta definición, en este caso: 640 x 480. La longitud de línea es de 640 pixeles, para nuestro algoritmo necesitamos 2 x 640 +3=1283 flip-flops por cada primitiva morfológica.

En los FPGAs se dispone de un flip-flop por cada LUT (Look-UP Table), este flip-flop se tiene para dar la opción de implementar salidas registradas. En el FPGA XCS100 (usado en este trabajo) se dispone de 1200 LUTs, esto nos da 1200 flip-flops disponibles para implementar diversos tipos de registros. Como puede verse, el total de los flip-flops en el FPGA no es suficiente para implementar una primitiva morfológica.

Buscando alternativas entre diversos fabricantes de FPGAs, se encontró que el fabricante Xilinx predispone en la arquitectura de sus FPGAs la posibilidad de implementar registros de corrimiento de gran tamaño. Dispositivos como SPARTAN II, SPARTAN III y VIRTEX, ofrecen la posibilidad de implementar registros de corrimiento sin agotar sus flip-flops, para esto se usan las LUT, con lo que se puede utilizar en su totalidad los recursos del FPGA.

EL software de desarrollo proporcionado por el fabricante incluye varias librerías [26] de circuitos, entre estas se encuentran las primitivas para los registros de corrimiento de 16 bits; implementados a través de las LUTs.

Usando la primitiva del registro de corrimiento de 16 bits se construye la macro para un registro de corrimiento de 640 bits, este registro se integra por una cascada de 40 registros de 16 bits, con el registro de 640 bits se puede almacenar una línea de pixels. La macro del registro de 640 bits se puede invocar cuantas veces se requiera.

Con la macro del registro de 640 bits y flip-flops discretos se conforma la arquitectura propuesta para la implementación de las primitivas de erosión y dilatación.

5.2 Implementación.

5.2.1 Conversión de RGB a Blanco y Negro

La señal de video requerida en este trabajo es en blanco y negro, por esta razón se requiere convertir la señal de entrada en formato RGB a blanco y negro.

El decodificador de video DVI entrega las siguientes señales:

- *HSYNC*: Sincronía Horizontal, marca el inicio de la línea
- *VSYNC*: Sincronía Vertical, marca el inicio del cuadro
- *CLK*: Pixel clock, entrega un pulso por cada píxel
- *DE*: Habilitación de datos, activa cuando hay datos
- *R0-R7*: 8 bits para rojo, valor de intensidad
- *G0-G7*: 8 bits para verde
- *B0-B7*: 8 bits para azul

Como puede verse el decodificador DVI entrega la señal de video en formato digital RGB, con 8 bits por color. Previo a la conversión de color a blanco y negro se realiza el paso intermedio de conversión de color a tonos de gris.

Para convertir la señal de color en tonos de gris existen varios algoritmos presentes en la bibliografía, de todos estos escogemos el más sencillo: promedio de los valores RGB.

Realizar un promedio requiere de las operaciones de adición y división, la implementación de la adición presenta su mayor retardo en la propagación del carry, la división sin duda es la operación que consume más tiempo y recursos de hardware.

En el criterio adoptado se realiza únicamente la suma de RGB, sin realizar división alguna. Para convertir a blanco y negro se realiza lo que se conoce como umbralización, la cual consiste en comparar el nivel de tono de gris contra un valor de referencia o umbral, en caso de ser menor se asigna cero, en caso de ser mayor se asigna uno. Considerando que el nivel de gris es tres veces mayor (no se hizo división) el valor de umbral a utilizar también es tres veces mayor, de esta manera se realiza la conversión de color a blanco y negro sin hacer uso de la división.

En la figura 5.2 se tiene el sumador RGB, consiste de dos sumadores de 16 bits conectados en cascada para sumar los tres datos del RGB, los bits de entrada no utilizados se conectan a cero (gnd).

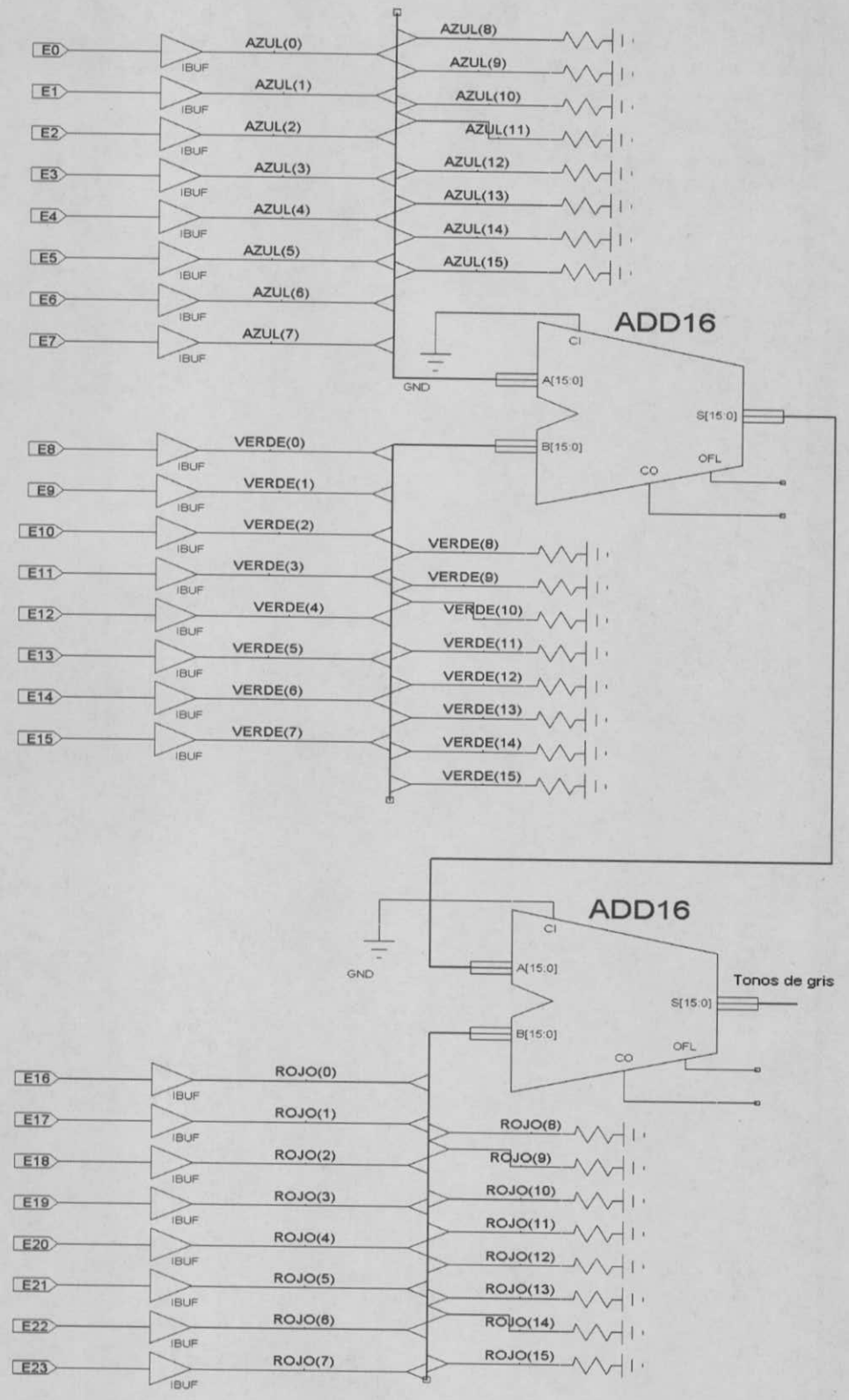


Figura 5.2: Sumador RGB

En la figura 5.3 se tiene el comparador de 16 bits usado en la conversión de tonos de gris a blanco y negro, en una entrada se conecta la salida del sumador, esta es el valor de intensidad del tono de gris, en otra entrada se conecta el valor de umbral, para poder modificar el valor de umbral los bits 6, 7, 8, 9, se conectan externamente a micro interruptores, para poder conmutar su valor entre uno y cero.

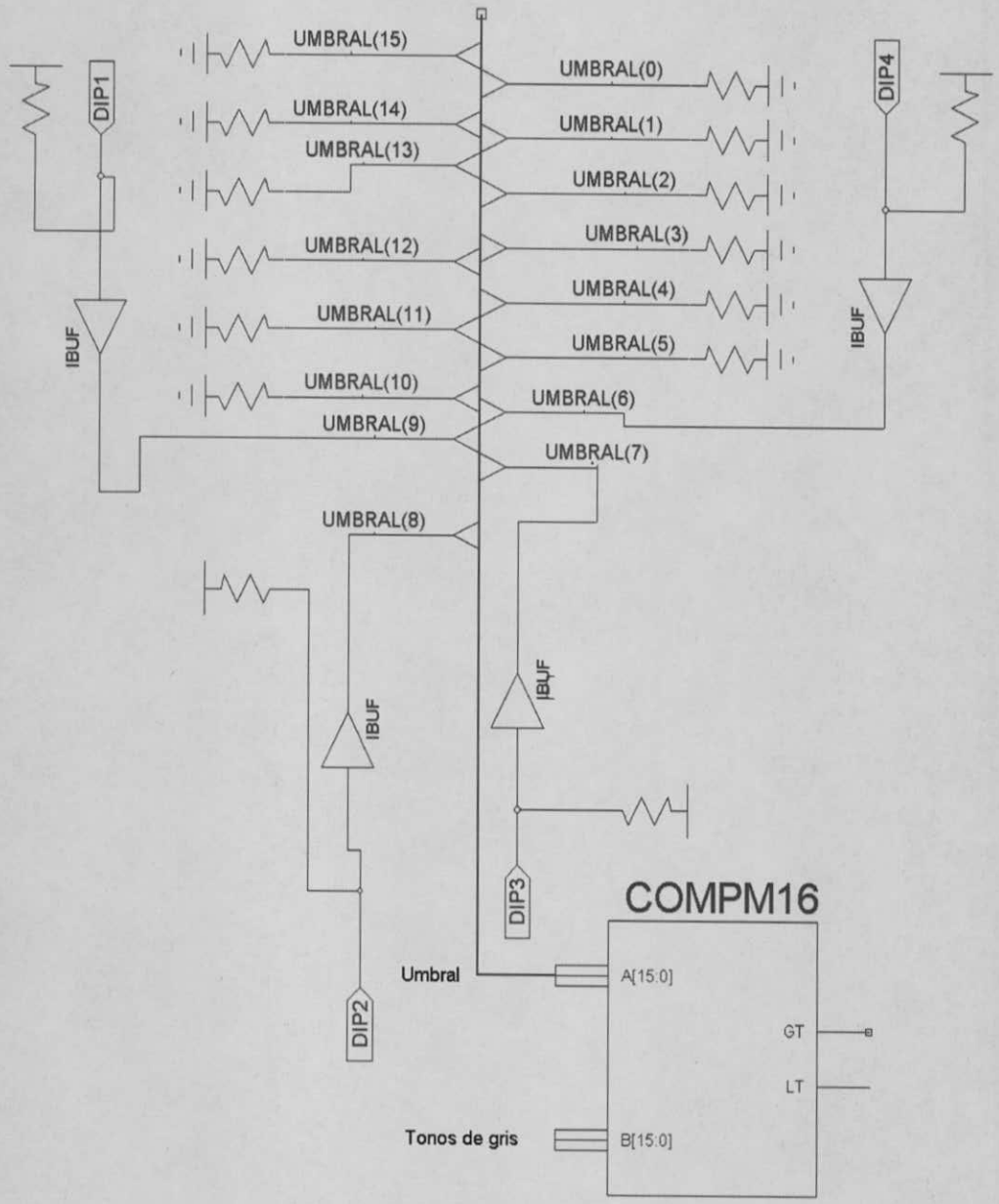


Figura 5.3: Comparador para umbralización

5.2.2 Primitivas Morfológicas

Registro 16 bits

Las primitivas para la erosión y la dilatación tienen como elemento formativo el registro de corrimiento de 16 bits, implementado a través de las LUT, la figura 5.4 muestra el diagrama de este registro, esta primitiva se encuentra disponible en las librerías de Xilinx, cuenta con cuatro líneas de selección *A0-A3,* usadas para determinar el tamaño del registro que puede ir de un bit (0000) a 16 bits (1111), entrada de datos *D,* señal de reloj *CK,* habilitación de reloj *CE* y salida *Q.*

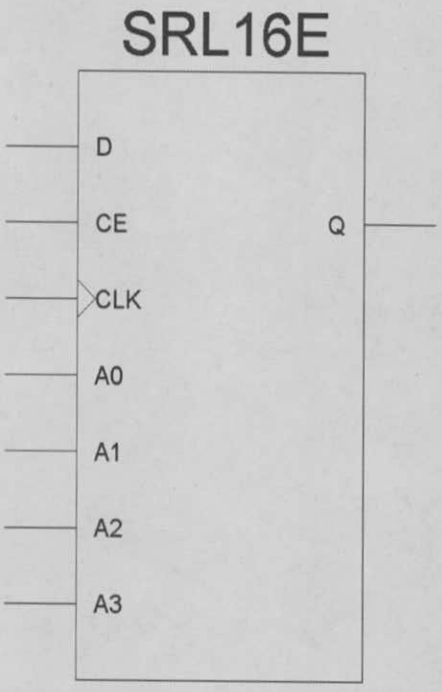


Figura 5.4: Registro de corrimiento 16 bits implementado con LUT

Para formar una línea se conectan en cascada varios registros de este tipo, para reducir el tamaño del circuito final se forman bloques previos como es el caso de un bloque de diez registros, ver figura 5.5.

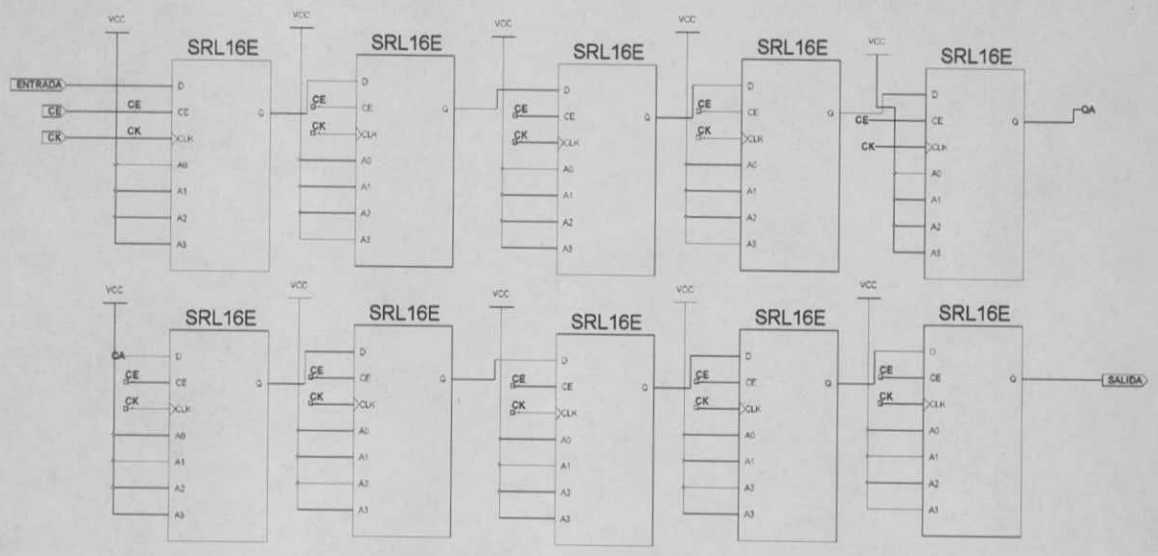


Figura 5.5: Bloque de diez registros de corrimiento de 16 bits

En la figura 5.7 se tiene un bloque similar de 10 registros, pero el último de estos está configurado con una longitud de 12 bits, los 4 bits faltantes se implementan con un registro de 4 bits el cual cuenta con salida para cada uno de los bits, sobre estos se aplica la operación lógica de la AND (erosión) y OR (dilatación).

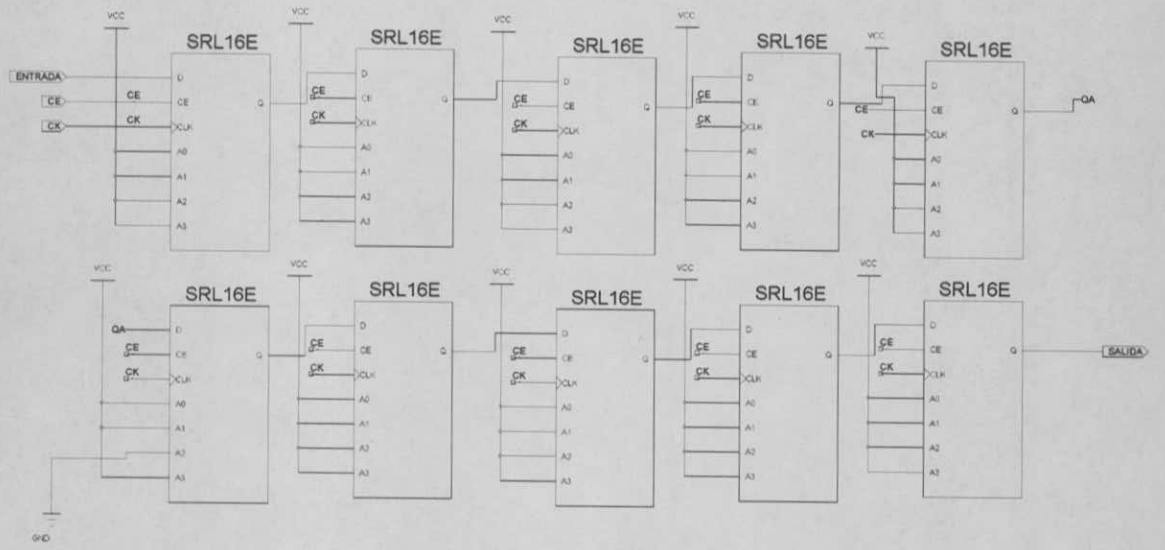


Figura 5.6: Bloque de diez registros de corrimiento, nueve con longitud de 16bits, uno con longitud de 12 bits

En la figura 5.7 se muestra el bloque de 4 bits usado al final de la línea, en este bloque se dispone de las salidas paralelas del registro, sobre éstas se aplica la OR y la AND.

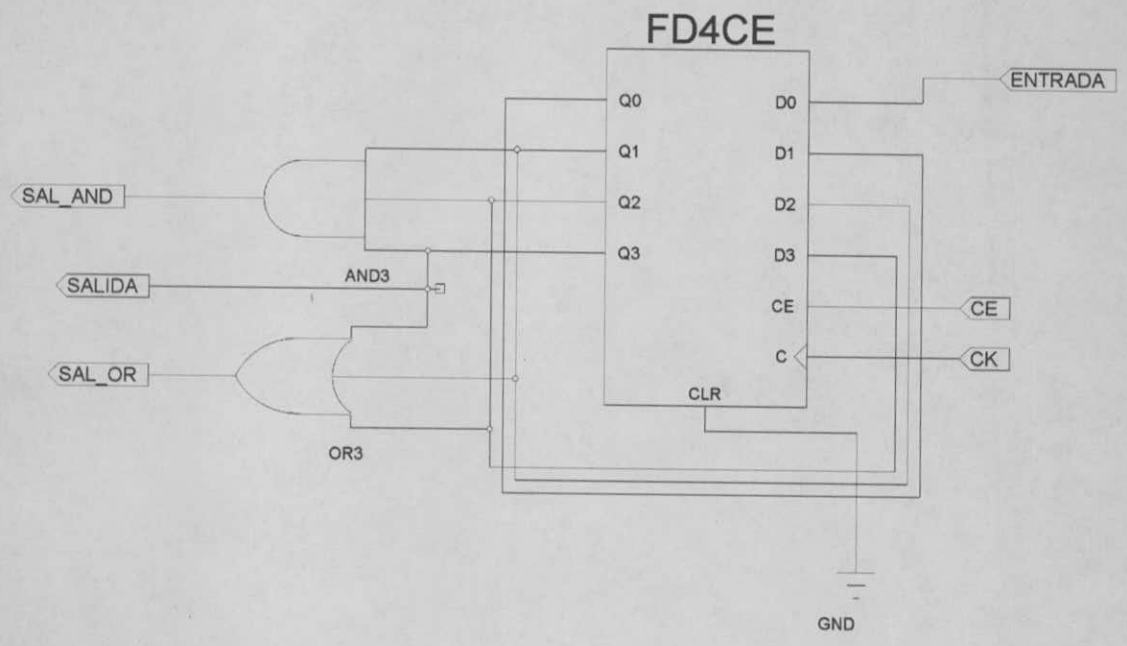


Figura 5.7: Bloque al final de la línea

Línea de 640 bits

La figura 5.8 muestra los elementos que integran una línea completa para una resolución de 640 pixels, de izquierda a derecha se tiene:

- Compuertas AND y OR conectadas a las salidas del bloque de cuatro bits.
- Bloque de 20 registros, en este bloque el último tiene una longitud de 12 bits.
- Bloque de 16 bits, este bloque se agrega por en realidad la línea de 640 tiene 8 espacios sin información de cada lado de la línea.
- Bloque de 20 registros (320 bits).

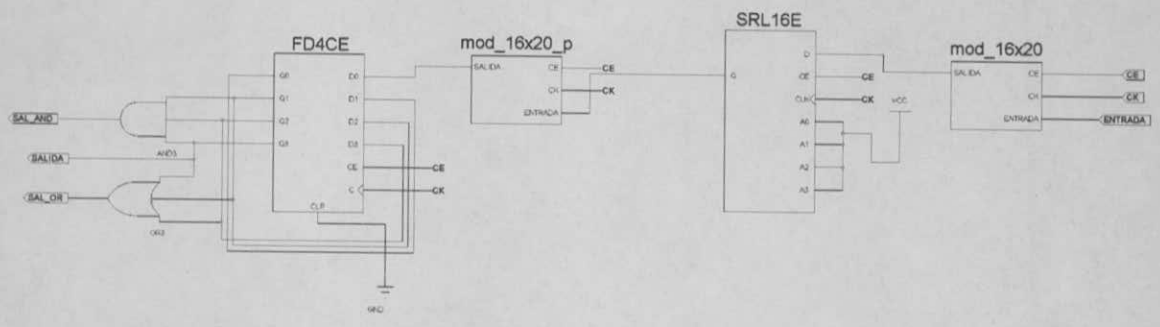


Figura 5.8: Línea de 640

Retardo

El retardo consiste en una línea de registros cuya longitud corresponde a la cantidad de pixels que se desean retrazar antes de aplicar alguna operación, para el caso de la operación de contorno se requiere un retardo con una longitud de una línea y tres puntos, el diagrama se muestra en la figura 5.9. En el diagrama se tiene un bloque de línea de 640 concatenado a un registro de 4 flip-flops "D", de estos se arreglan tres para formar un registro de corrimiento de tres bits.

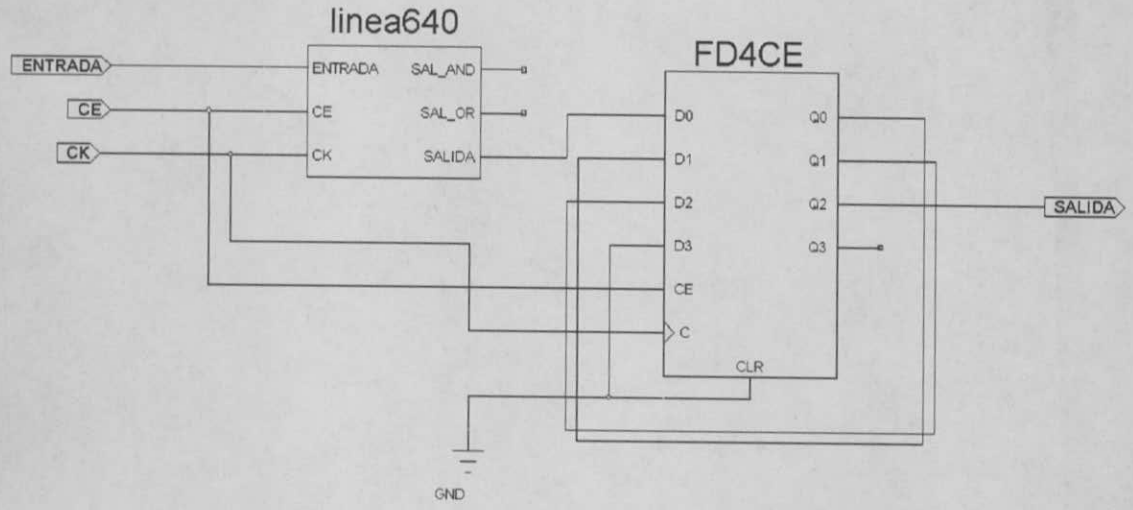


Figura 5.9: Línea de retardo

Primitiva de Erosión y Dilatación

La primitiva básica de erosión y dilatación se muestra en la figura 5.10, esta primitiva es fielmente la implementación de la arquitectura propuesta, se forma con dos líneas de 640 bits, un segmento de línea de cuatro bits, compuertas lógicas AND (erosión) y OR (dilatación).

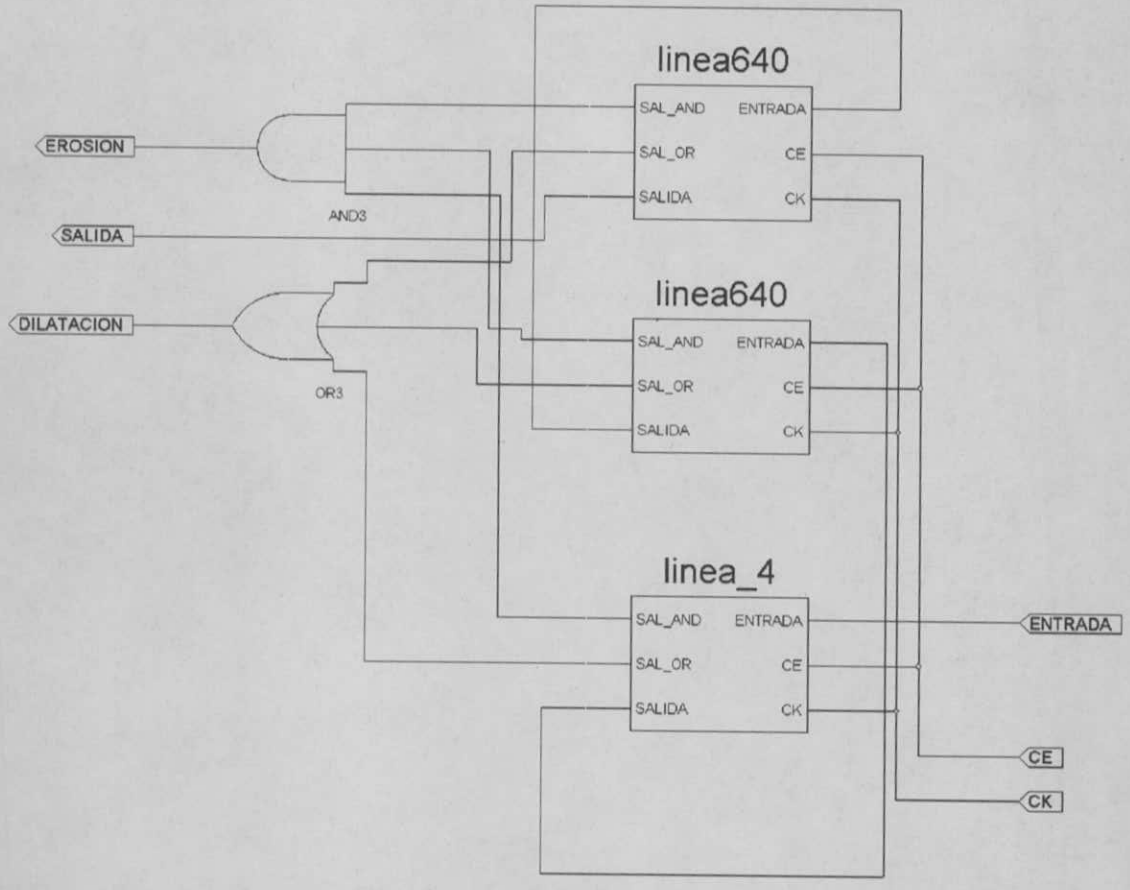


Figura 5.10: Primitiva de Erosión y Dilatación

Primitiva para la extracción del contorno

La primitiva para el contorno se muestra en la figura 5.11, se integra por la primitiva de erosión-dilatación, un retardo, un inversor y una OR exclusiva; estos elementos corresponden con la ecuación del contorno: $\Gamma(A) = A - (A \Theta B)$.

Figura 5.11: Primitiva para el contorno

5.2.3 Software de desarrollo

En la figura 5.12 se muestra el entorno de trabajo de Xilinx, a través de este entorno se crean los archivos fuente, por medio de lenguajes de descripción de hardware, diagramas de estado y esquemáticos. Como se ha mostrado este sistema se realizó por medio de esquemáticos, dado que el diseño se adapta de manera natural (registros de corrimiento) a la descripción esquemática.

Trabajar con descripción esquemática proporciona mayor eficiencia en cuanto a recursos del FPGA se refiere, situación comparable a lenguajes de alto nivel y lenguaje ensamblador, en ensamblador se trabaja directamente con instrucciones del microprocesador.

Los procesos de Síntesis e Implementación se realizan desde este mismo entorno, en la figura 5.12 se tiene en la parte izquierda superior la ventana donde administran todos los

archivos fuente, en la parte izquierda inferior está la ventana de procesos (síntesis e implementación) para los archivos fuente.

En la ventana de la derecha se presentan reportes de los diferentes procesos; en este caso se tiene parte del reporte de la síntesis, se puede observar la tabla que reporta el porcentaje de uso de los recursos del FPGA. Para la conversión de color a B&N y la operación de contorno en la tabla se reporta un uso de 89 Slices de un total de 1200, esto equivale al 7% de los recursos del FPGA, restando el 93% de los recursos del FPGA para implementar más primitivas morfológicas. Cada primitiva consume aproximadamente el 2% de los recursos del FPGA, por lo que sin problema se puede implementar un promedio de 40 primitivas.

En la figura 5.13 se tiene el editor de esquemáticos, con este editor y las primitivas disponibles en las librerías, se diseñan los circuitos digitales. La creación de macros a partir de otros esquemáticos o por lenguajes de descripción de hardware, extiende la variedad de esquemas disponibles.

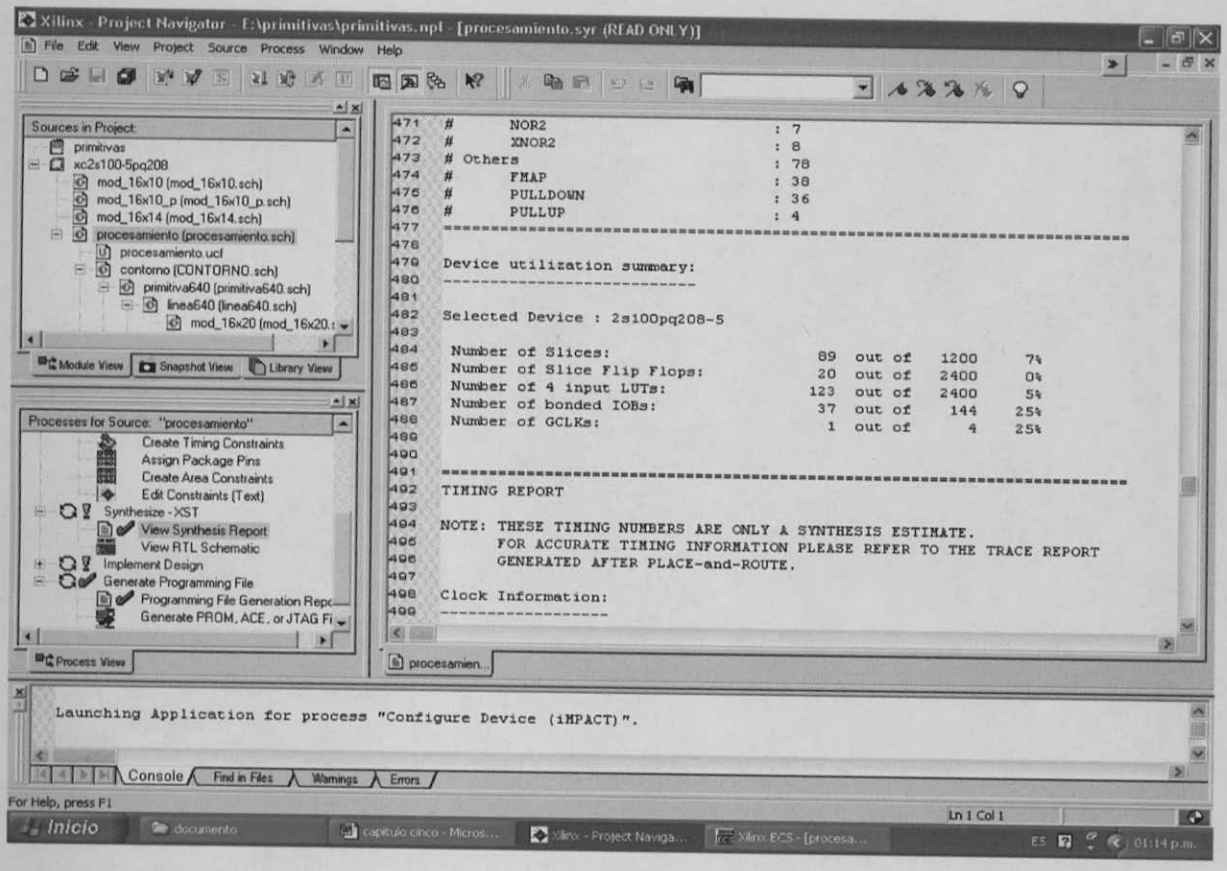


Figura 5.12: Entorno de trabajo.

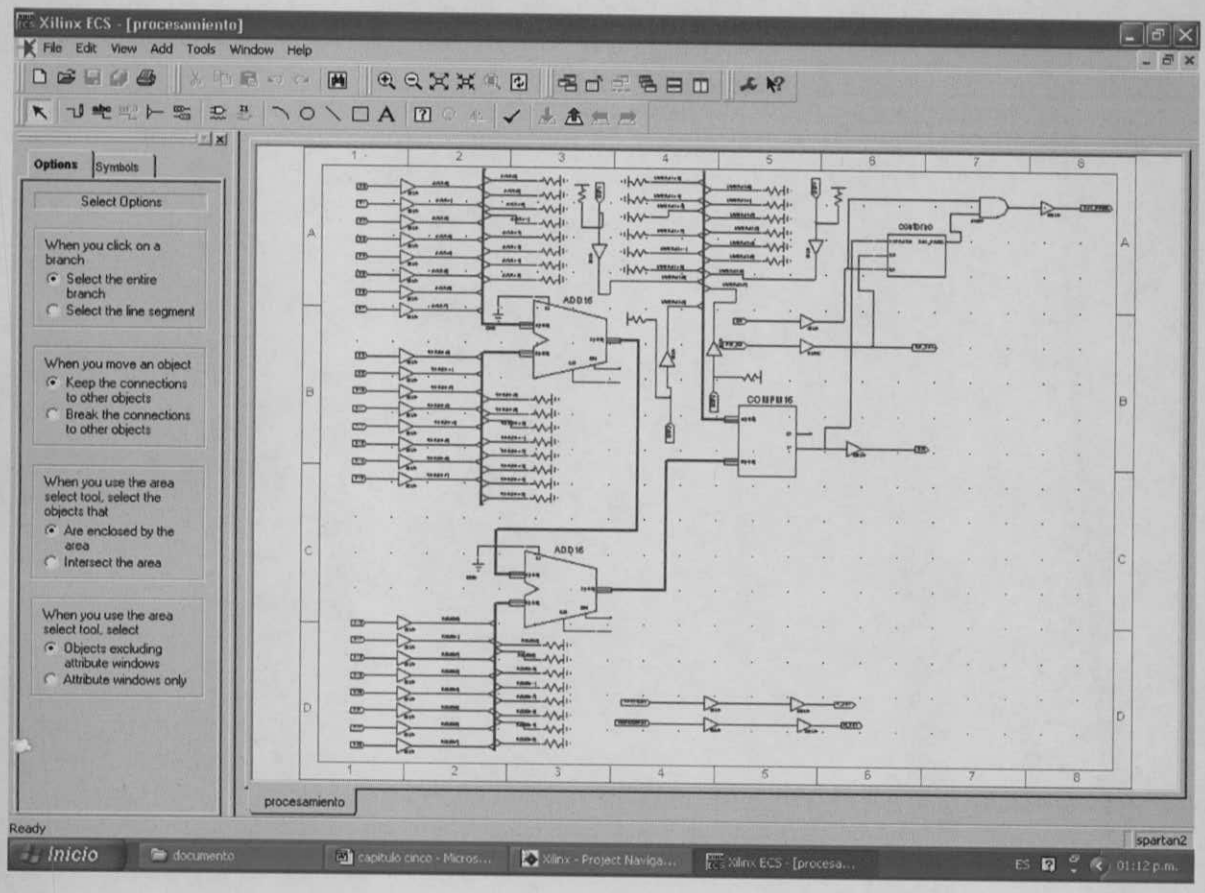


Figura 5.13: Editor de esquemáticos

En nuestro caso se han creado macros para las primitivas morfológicas de erosión, dilatación y retardo, ahora estas primitivas forman parte de los recursos disponibles en el editor de esquemas. El editor de esquemas nos sirve como sistema didáctico y de desarrollo para aplicaciones de Morfología Matemática.

5.2.4 Tarjeta decodificadora DVI

Para la decodificación de las señales DVI se utiliza la tarjeta de la figura 5.14; ésta se integra principalmente por el C.I. SIL161BCT100, circuito que se encarga de la decodificación de las señales del formato DVI.

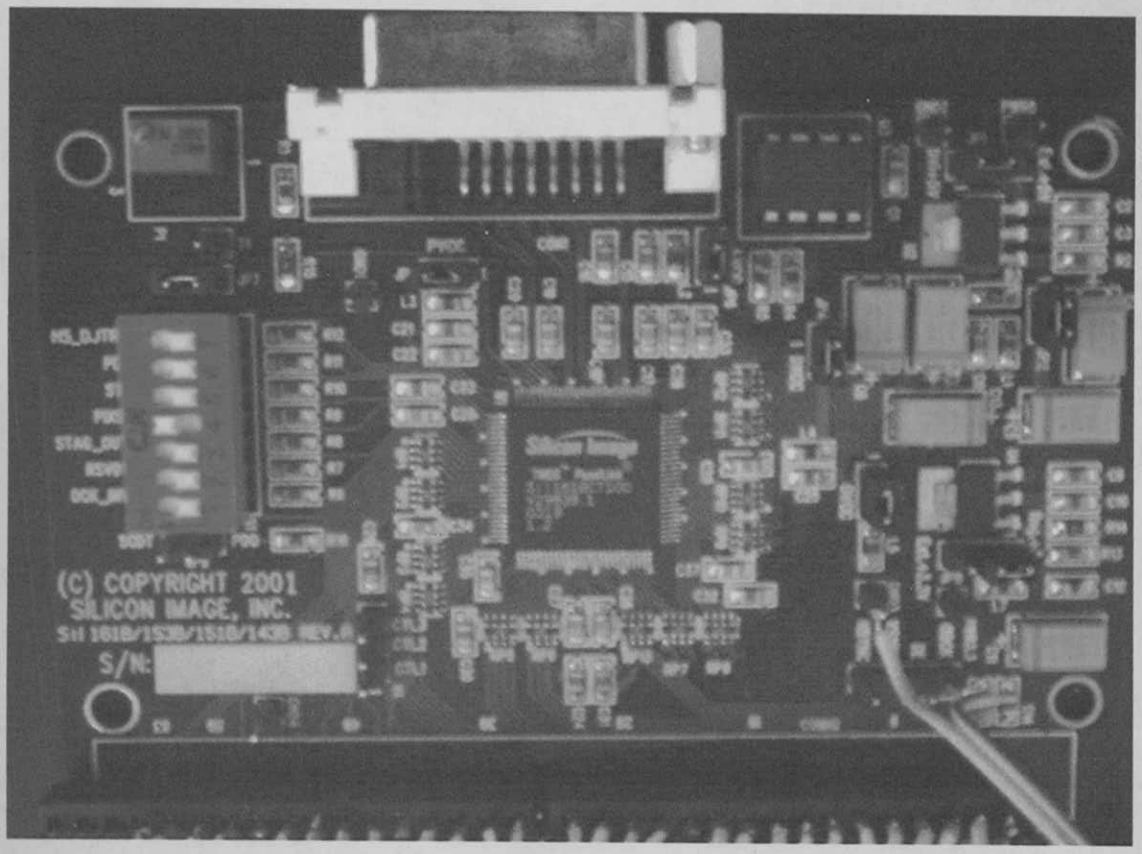


Figura 5.14: Tarjeta para decodificación de DVI

La tarjeta decodificadora entrega las señales *(HSYNC)* Sincronía Horizontal, *(VSYNC)* Sincronía Vertical, *(CLK)* Píxel clock, *(DE)* Habilitación de datos, *(R0-R7)* 8 bits para rojo, *(G0-G7)* 8 bits para verde, *(B0-B7)* 8 bits para azul.

En la figura 5.15 se tiene el oscilograma del pixel clock, su frecuencia es de 25 MHz. En la figura 5.16 se presenta el oscilograma de la señal de habilitación de datos y de la sincronía horizontal, en este oscilograma se aprecia claramente que ambas señales están en fase, la diferencia se tiene en la duración de la parte alta de las señales, *DE* tiene menor duración, esto nos indica que en los extremos de la línea de datos hay espacios vacios.

Figura 5.15: Señal de píxel clock a 25 MHz.

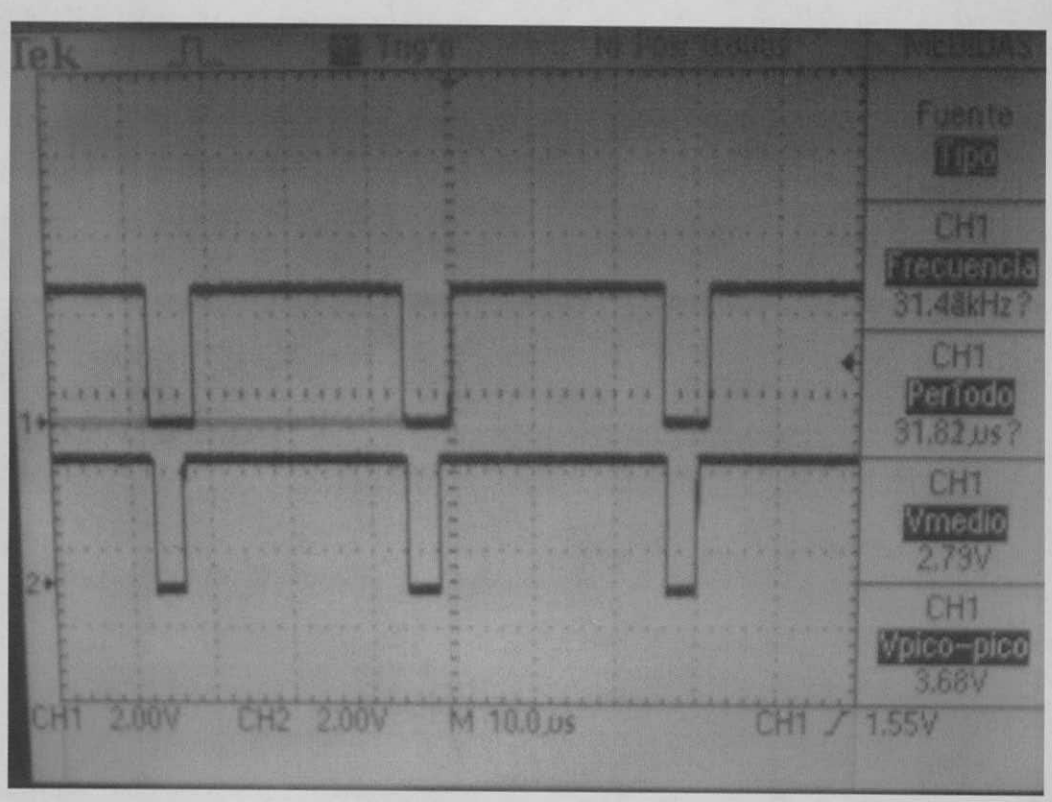


Figura 5.16: Señal de Habilitación de datos y sincronía horizontal.

En la figura 5.17 se tiene la tarjeta de procesamiento integrada por el FPGA SPARTAN II XC2S100, a través del cual se implementa la arquitectura presentada en este trabajo. La tarjeta recibe las señales del decodificador DVI, la señal se procesa y se acondiciona para una salida analógica.

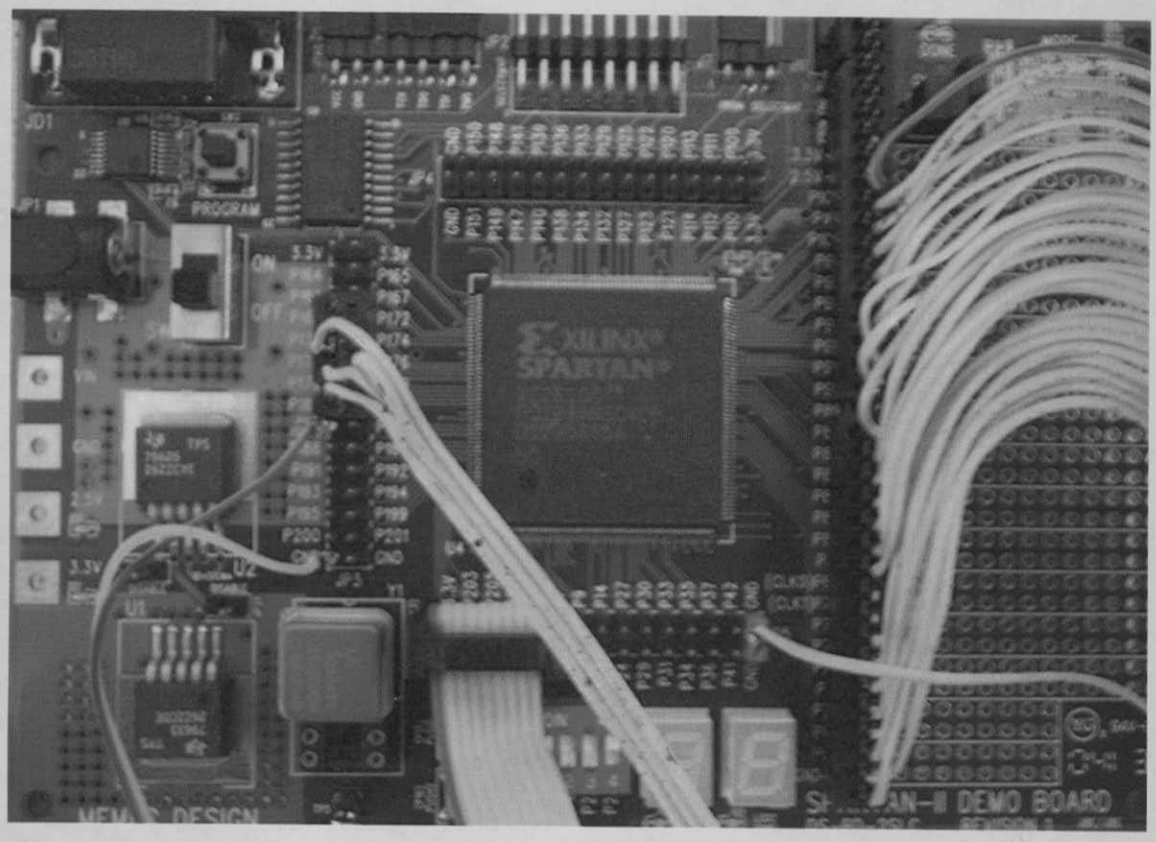


Figura 5.17: Tarjeta con el FPGA SPARTAN II XC2S100.

5.2.5 Organización del sistema

El sistema se soporta en dos computadores y cuatro monitores: el primer computador tiene instalada la tarjeta de video con salida analógica y salida digital (DVI), la salida DVI se conecta a la tarjeta decodificadora. En este computador se genera la imagen fuente configurada a una resolución de 640 x 480 a 60 cuadros por segundo.

El segundo computador tiene instalado el software de desarrollo a través de cual se editan, sintetizan e implementan las primitivas morfológicas deseadas. Con las macros de las primitivas morfológicas y el software de desarrollo se tiene una plataforma de desarrollo

para cuestiones didácticas y aplicaciones de la Morfología Matemática.

El puerto paralelo se usa para descargar el archivo del diseño al FPGA, éste es de tecnología reprogramable, por lo tanto se puede modificar el diseño original o bien se puede agregar un diseño nuevo.

De los cuatro monitores, dos se utilizan en los computadores, el tercero se conecta a la salida de B&N, el cuarto va conectado a la salida de la señal procesada. En la figura 5.18 se representa a bloques la organización del sistema.

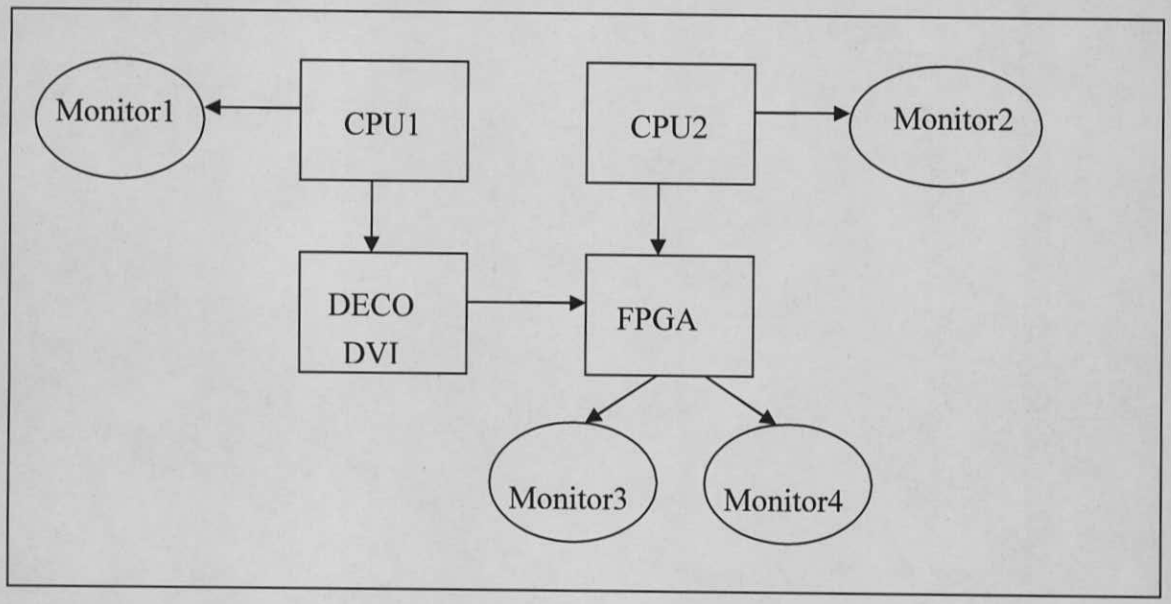


Figura 5.18: Organización del sistema

5.2.6 Ejemplo de aplicación: extracción de contornos

La extracción de contornos es la operación morfológica elegida para dar una muestra de los resultados obtenidos en este trabajo. En las figuras 5.19 y 5.20 se tienen ejemplos de la extracción de contornos. La imagen (a) es la imagen original, esta se presenta en color, con una resolución de 640 x 480 píxeles a 60 cuadros por segundo. La imagen (b) es la conversión a B&N de la imagen original en color, mantiene la resolución y número de cuadros por segundo de la imagen original. En la imagen (c) se tiene la imagen procesada, en este caso de aplicación se tiene la extracción del contorno de la imagen en B&N presentada en (b).

Las imágenes en (a), (b) y (c) son fotos tomadas directamente a los monitores 1, 3, 4, el ruido presente en las imágenes se puede atribuir a la diferencia de barrido entre los monitores y la cámara fotográfica.

(a)

(b)

(c)

Figura 5.19: Imagen en color (a), blanco y negro (b), contorno (c).

(a)

(b)

Figura 5.20: Imagen en color (a), blanco y negro (b), contorno (c).

Conclusiones y trabajo futuro

Conclusiones

Con el algoritmo y la arquitectura desarrollada en este trabajo de tesis, se demuestra que el retardo obtenido para un elemento estructurante de 3x3; es equivalente a la duración de *una línea de video y dos pixeles,* el cual es el mínimo retardo posible desde el punto de vista algorítmico, y la arquitectura específica desarrollada lo ha hecho posible. Al procesar la imagen mientras fluye (en forma de señal) se evita cualquier otro tipo de retardo, desde que entra la imagen y sale procesada.

En referencia a trabajos relacionados, revisados en esta tesis, no es posible hacer una comparación directa, dado que la mayoría de estos trabajos están pensados para el procesamiento de imágenes en general, y no en específico como ocurre en este trabajo.

De los trabajos revisados, el que reporta los mejores tiempos de procesamiento para la operación de dilatación, es el sistema de IMAP-VISION [3]. Para una imagen de 256 x 256 pixeles, el sistema de IMAP-VISION reporta un tiempo de procesamiento de 0.31ms (310μs).

Para una operación de dilatación o erosión, nuestro sistema requiere de un tiempo de procesamiento igual a la duración de una línea de video y dos pixeles. Para una imagen de 640 x 480 a 60Hz, la línea de video tiene una duración de 30.862 μs, la cual se integra por: 640 pixeles activos (25.453 μs), 8 pixeles del Front porch (0.318 μs), 96 pixeles del Sync width (3.818 μs) y 32 pixeles para el Back porch (1.273 μs).

El tiempo de procesamiento es igual a: Línea completa de video + sync width + front porch + dos pixeles (30.862 μs + 3.818 μs + 0.318 μs + 0.0795) = **35.0775 μs.** El tiempo de procesamiento es aproximadamente 9 veces menor que el obtenido por el sistema IMAP-VISION.

Al utilizar las LUTs (Look-Up Table) del FPGA como registros de corrimiento se logró aprovechar los recursos del FPGA al cien por ciento.

El FPGA empleado tiene una densidad de 100,000 compuertas y una frecuencia de sistema de 200Mhz. Con estos recursos del FPGA se puede implementar un promedio de 40

primitivas morfológicas y procesar satisfactoriamente imágenes con una resolución de 640 x 480 a 60 cuadros por segundo.

Con las macros desarrolladas para las primitivas morfológicas y el editor de esquemas se integró un sistema de desarrollo flexible para cuestiones didácticas y diversas aplicaciones de la Morfología Matemática.

Las cámaras inteligentes tienen integrado un sistema de procesamiento, con el cual pueden evaluar alguna tarea específica. El sistema desarrollado se implementa en un chip, en éste se puede integrar un algoritmo completo para una tarea específica, con la finalidad de construir un sistema para cámaras inteligentes.

El costo total del sistema está por debajo de los 100 USD, la tarjeta de evaluación del FPGA tiene un costo de 50 USD, el circuito decodificador de DVI (SIL161BCT100) cuesta 12 USD y una tarjeta de video con salida DVI se puede adquirir por 35 USD. El software de desarrollo es de distribución libre.

Trabajo futuro.

En la morfología matemática existen algoritmos iterativos, estos algoritmos exigen la imagen completa para iterar sobre ésta hasta que se cumpla cierta condición. Un trabajo inmediato es agregar al sistema memoria FIFO externa para poder ejecutar los algoritmos iterativos. Para estos algoritmos se puede ver reducido el número de cuadros procesados por segundo, pero a pesar de esto el uso de la arquitectura específica es una mejor opción que usar DSPs.

Aumentar la resolución de las imágenes, implica aumentar la frecuencia de transmisión de las señales, por ejemplo: para una resolución de 1024 x 768 la señal de reloj alcanza una frecuencia de 65 MHz. Usar un FPGA de mayor grado de velocidad puede servir para este propósito.

Para imágenes en tonos de gris se requieren ocho bits para representar un pixel; siete más que para blanco y negro, aumentando la densidad de compuertas en la misma proporción, este trabajo se puede llevar a tonos de gris. Para esto se puede utilizar un FPGA Spartan-3 de 1000k compuertas.

Referencias Bibliográficas

Referencias bibliográficas

[1] Thomas Bräunl, Tutorial in Data Parallel Image Processing, Dept. of Electrical and Electronic Engineering, Centre of Intelligent Information Processing Systems,The University of Western Australia, 2001

[2] Thomas Bräunl, Parallaxis-III, User Manual, Computer Science Report, Bericht Nr. June 1996

[3] Stelian Persa, CristinaNicolescu, Pieter Jonker, Evaluation of Two Real Time Low Level Image Processing Architectures, Pattern Recognition Group, Delft Technical University, The Netherlands, 2000.

[4] Molen, van der, M.W., Kyo, S., "Documentation for the IMAP-VISION image processing card and the 1DC language", NEC incubation Center, 1997

[5] T.G. Morris and S.P. DeWeerth, An analogue VLSI morphological image processing circuit. *Electronics Letters,* November 1995.

[6] Drayer, Thomas Hudson, A Design Methodology for Creating Programmable Logic-based Real-time Image Processing Hardware, Department Electrical and Computer Engineering, Blacksburg, Virginia, 1997-01-24

[7] Steffen Klupsch and Markus Ernst and Sorin A. Huss and Martin Rumpf and Robert Strzodka, Real Time Image Processing based on Reconfigurable Hardware Acceleration, Proceedings of IEEE Workshop Heterogeneous reconfigurable Systems on Chip, 2002.

[8] James Fung, Steve Mann, Chris Aimone, "OpenVIDIA: Parallel GPU Computer Vision", *Proceedings of the ACM Multimedia 2005* , Singapore, Nov. 6-11, 2005, pages 849-852

[9] Matthias Hopf and Thomas Ertl, Accelerating Morphological Analysis with Graphics Hardware, Workshop on Vision, Modelling, and Visualization 2000, pp 337-345).

[10] Antony Rowstron and Alan Wood, Implementing Mathematical Morphology in ISETL-LINDA, Proceedings of the 5th International Conference on Image Processing, pp847-851, IEE, 1995.

[11] PUJOL, F.A., GARCÍA, J.M., LEDESMA, B., FUSTER, A., PUJOL, M., "Filtrado morfológico de imágenes: implementación de primitivas basada en DSPs", VIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'99), Murcia, noviembre 1999.

[12] F. Robin, M. Renaudin, G. Privat, "An Asynchronous 16*16 Pixel Array-Processor for Morphological Filtering of Greyscale Images", Proceedings of the European Solid-State Circuits Conference (ESSCIRC'96), Neuchâtel, Switzerland, September 1996, pp. 188-191

[13] R. Spencer and E. Sanchez-Sinencio, "Monolithic mixed-mode implementation of sum-of-product arrays for performing morphological image processing," 1997 IEEE Int'l Symposium of Circuits and Systems (ISCAS), Hong Kong, Vol. 2, pp. 1421-4, 1997

[14] Díaz de León Santiago J., Yáñez Márquez C. (2003), *Introducción a la morfología matemática de conjuntos,* IPN, UNAM, FCE, México, D.F.

[15] Dougherty Edward R and Lotufo Roberto A. (2003), Hands-on Morphological Image Proccessing, SPIE Press.

[16] González R. and Woods R. (1993), *Digital Image Processing,* Addison-Wesley.

[17] Pajares G. y de la Cruz J. (2002), *Visión por computador*, Alfaomega Ra-Ma, México, D.F.

[18] http://www.ph.tn.tudelft.nl/Courses/FIP/frames/fip.html,Tutorial en línea sobre Morfología Matemática. Contiene definiciones de las operaciones con ejemplos numéricos y gráficos.

[19] Digital Display Working Group (1999). Digital Visual Interface revision 1.0

[20] Spartan-II 2.5V FPGA Family: Complete Data Sheet, Xilinx

[21] TFP101, TFP101A Design Notes, Texas Instruments.

[22] SiI161B PanelLink Receiver Datasheet, 2001, Silicon Image, Inc.

[23] TFP401, TFP401A, TI PanelBus DIGITAL RECEIVER, Texas Instruments.

[24] Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Generation FPGAs, Xilinx

[25] VESA ENHANCED EXTENDED DISPLAY IDENTIFICATION DATA, Implementation Guide Version 1.0, 2001, Video Electronics Standards Association

[26] Libraries Guide, Xilinx

[27] Silicon Graphics, Inc. (1997), *OpenGL Programing Guide*, Addison – Wesley Publishing Company.