PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UNA ARQUITECTURA PARA UN FILTRO MORFOLÓGICO DE IMÁGENES DIGITALES EN ESCALA DE GRISES EN UN FPGA DE ALTERA

Tesis para Optar el Título de:

INGENIERO ELECTRÓNICO

Presentado por:

JORDÁN GIACOMO VITELLA ESPINOZA

Lima – Perú

2003

RESUMEN

Los Filtros Morfológicos (MF) emplean operaciones no lineales sobre vecindades [1] [2] de píxeles en una imagen, que tienen como finalidad resaltar o remover selectivamente las estructuras u objetos presentes en ella; todos parten de dos operaciones básicas que son la dilación y la erosión, que se basan en la unión e intersección de conjuntos [1] respectivamente. Debido a que los algoritmos de los MF requieren procesar gran cantidad de datos a altas velocidades es necesaria su implementación en procesadores de aplicación específica (*hardware*). Este trabajo presenta el diseño e implementación de una arquitectura flexible que realiza las operaciones básicas de filtrado morfológico (dilación y erosión) en imágenes en escala de grises, aprovechando el paralelismo brindado por los arreglos de puertas programables por campo (FPGA). El diseño se basa en una arquitectura *pipeline* que permite separar el proceso de una imagen en varias etapas de procesamiento, la cual se implementó usando el lenguaje de descripción de *hardware* VHDL. Se realiza un análisis del rendimiento de sus diferentes partes (máxima frecuencia de reloj posible y consumo de celdas lógicas), según el tamaño de las imágenes para diferentes familias de FPGAs. Estos análisis fueron realizados tanto en el entorno del MAX+plus II como del Quartus II considerando las diferentes opciones de síntesis lógicas.

Los resultados de las pruebas experimentales realizadas demuestran una alta confiabilidad en los resultados y buenas velocidades de procesamiento, pudiendo procesar imágenes en escala de grises de 256 tonos con un tamaño de 256x256 píxeles, utilizando un elemento estructural de 3x3 píxeles en la misma escala de grises, a una velocidad de 58.47MHz con una ocupación del 42% del FPGA FLEX10K100EQC240-3.

A mis padres

Al Ing. Akio Yoshimoto, por haber sacrificado su tiempo en leer cada una de las revisiones de esta tesis, y por sus acertados consejos y recomendaciones que guiaron el desarrollo de esta

A Manuel Morales, por todos los sabios consejos brindados para la realización del presente trabajo

Al grupo de procesamiento digital de señales e imágenes (GPDSI) por el apoyo intelectual y las facilidades que me brindaron, por todo lo aprendido y lo vivido

Andres, Danielux, Gerard, Vico, Pedro, Jorge

A los amigos que me dieron ánimos y mostraron su preocupación.

Claudia, José, Dorelli

A todos ellos, mi gratitud.

“La persona que tiene intactos sus sueños, nunca está sola ni acabada. Sus ojos buscan la oportunidad, sus oídos reciben la dirección, su mente precisa un desafío y su corazón anhela el camino de Dios. Todas las personas de acción son primero soñadores.”

John Mason.

ÍNDICE

Pág.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La Morfología Matemática [1] es definida como una teoría para el análisis de estructuras espaciales; se la llama morfología porque aspira al análisis de las formas de los objetos y es matemática en el sentido que su análisis está basado en la teoría de conjuntos. En las áreas de procesamiento digital de imágenes y visión artificial, ha tenido un crecimiento acelerado debido a la variedad de aplicaciones que brinda. Aunque inicialmente fue desarrollada como una herramienta para analizar formas geométricas en imágenes binarias, la Morfología Matemática se ha convertido en un instrumento muy útil en el tratamiento de imágenes en escala de grises, que emplean operadores capaces de manejar sofisticados procesos de aplicación general a imágenes.

Los Filtros Morfológicos (MF) no son lineales porque no cumplen con el principio de superposición, además los métodos que emplea usan pequeñas vecindades [1] [2] de píxeles a una imagen de entrada para producir un nuevo valor de intensidad luminosa obteniéndose una imagen resultado. Por ello estos filtros no lineales son usados para resaltar o remover selectivamente estructuras u objetos de una imagen; con ello la selección se basa en la geometría y los contrastes locales de las formas o estructuras de estas, en este sentido un filtro morfológico es una interpretación de una imagen.

Los MF son comúnmente usados en pre-procesamiento de imágenes con dos objetivos: restauración de imágenes ruidosas (suavizado), y detección de bordes. El suavizado busca reducir el ruido u otras variaciones pequeñas en

una imagen, es equivalente a suprimir las frecuencias altas en el dominio de la transformada de Fourier. Por su parte, detectar los bordes tiene un efecto similar a suprimir las bajas frecuencias.

Las operaciones básicas dentro del Filtrado Morfológico son la Dilación y la Erosión [1]. La Dilación de una imagen binaria implica un enriquecimiento de la población de sus píxeles, mientras que la Erosión de la misma obtiene un empobrecimiento de su población. Estos procesos se basan en las operaciones de unión e intersección de la teoría de conjuntos respectivamente, con ello su aplicación y propiedades se extienden al caso de las imágenes en escala de grises; esto es, se añade una dimensión que corresponde a la intensidad luminosa en cada punto de la imagen. La operación de Dilación obtiene un crecimiento de la formas de la imagen pues expande las zonas claras, en cambio la operación de Erosión encoge las formas expandiendo las zonas oscuras. Ambas operaciones son logradas utilizando, además de la imagen a filtrar, una imagen pequeña conocida como Elemento Estructural (SE), cuya forma determina la vecindad [1] [2] de píxeles que es operada por el Filtro Morfológico. En las aplicaciones de imágenes en escala de grises, el SE puede ser plano (binario) o con volumen (en escala de grises), su forma y tamaño deben ser adaptados a las propiedades geométricas de la imagen en proceso [3].

Existe un amplio desarrollo de las operaciones morfológicas en lenguajes de programación (*software*), pero debido a la gran cantidad de datos contenidos en la mayoría de aplicaciones de visión por computador, se optó por el diseño

de arquitecturas para ser implementadas en procesadores de aplicación específica (*hardware)*. Previamente se ha investigado la implementación de operaciones morfológicas [4] en un FPGA [5], pero esto solo contempla el caso de imágenes monocromáticas.

El presente trabajo amplía esta rama de investigación para imágenes en escala de grises. La propuesta se basa en una arquitectura *pipeline* [6] [7] para la implementación de las operaciones básicas de filtrado morfológico en paralelo, la cual es presentada en el Capítulo 1. En el capítulo 2, se muestra la forma de implementar la arquitectura sugerida, y se realiza un estudio de su rendimiento sobre los FPGA de Altera, en las familias FLEX10K100E [8], ACEX1K [9] y Stratix [10], para imágenes con un tamaño máximo de 256x256 píxeles y 256 tonalidades de grises, con un elemento estructural (SE) de 3x3 píxeles que tiene las mismas tonalidades. La prueba del sistema fue realizada en la tarjeta de desarrollo Constellation-E de Nova Engineering [11], que permite la comunicación entre la PC y un FLEX10KE100EQC240-3 [8] a través del Bus ISA. Por ello el sistema implementado incluye una interfaz para el bus ISA, la cual permite un flujo de datos permanente entre la memoria de la PC y la arquitectura, de manera que se puedan cargar nuevos datos y descargar resultados simultáneamente. Los resultados teóricos alcanzados en las simulaciones y pruebas experimentales, así como su comparación con algoritmos implementados en Matlab [12], son mostrados en el Capítulo 3.

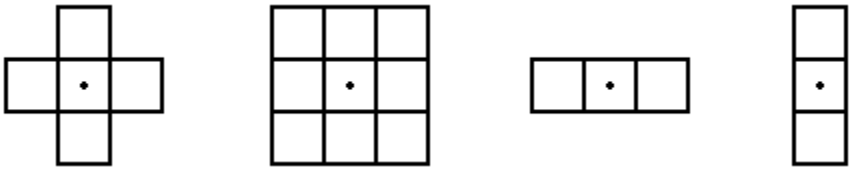
1. SELECCIÓN DE LA ARQUITECTURA Y EL ALGORITMO DE MORFOLOGÍA MATEMÁTICA A DESARROLLAR

1.1.- ANÁLISIS DEL FILTRADO MORFOLÓGICO

Un MF tiene como entradas la información de la imagen que se desea filtrar y otra imagen, comúnmente pequeña en comparación a la imagen a procesar, denominada Elemento Estructural (SE) [2]. La nueva imagen resultado de la aplicación de un MF es la imagen sometida a transformaciones en los valores de intensidad luminosa de cada uno de sus píxeles, estas transformaciones dependen de la luminosidad de cada píxel y la de sus vecinos [1] [2].

La información contextual de un píxel es dada por los valores enteros de los que conforman su vecindad [1] [2]. Aunque en teoría no hay limitación en el tamaño de una vecindad, la información contextual necesaria de un píxel es presentada en pequeñas vecindades. Por otro lado, los requerimientos computacionales crecen geométricamente cuando el tamaño de la vecindad aumenta.

El SE es el punto de partida de todas las operaciones morfológicas, puede tener cualquier forma y tamaño, que dependen de la aplicación que se le desee dar, sin embargo solo algunas formas son usadas en aplicaciones prácticas (ver figura 1.1).



(A) (B) (C) (D)

FIGURA 1.1.- FORMAS DE ELEMENTOS ESTRUCTURALES COMÚNMENTE USADOS

En el caso de imágenes binarias el SE solo determina la forma y tamaño de la vecindad de píxeles, mientras que para imágenes en escala de grises cada píxel del SE está asociado a un valor entero en la misma escala que la imagen, los cuales adicionan pesos [1] a los píxeles pertenecientes a la vecindad además de indicar su forma y tamaño.

Un proceso de filtrado de imágenes consiste en realizar una serie de operaciones sobre cada uno de los píxeles que la componen. Durante el proceso, el SE es desplazado a través de toda la imagen, su píxel central se posiciona sobre cada píxel de la imagen y su forma determina una vecindad. Por ejemplo, sea una imagen A la cual se filtra con un SE (ver figura 1.2) cuyo centro se ubica sobre la posición *(i, j)* de A, siendo *i* el número de una fila dada y *j* el número de una columna dada sobre la imagen; se obtiene la substitución del valor del píxel en la posición *(i, j)* por un nuevo valor.

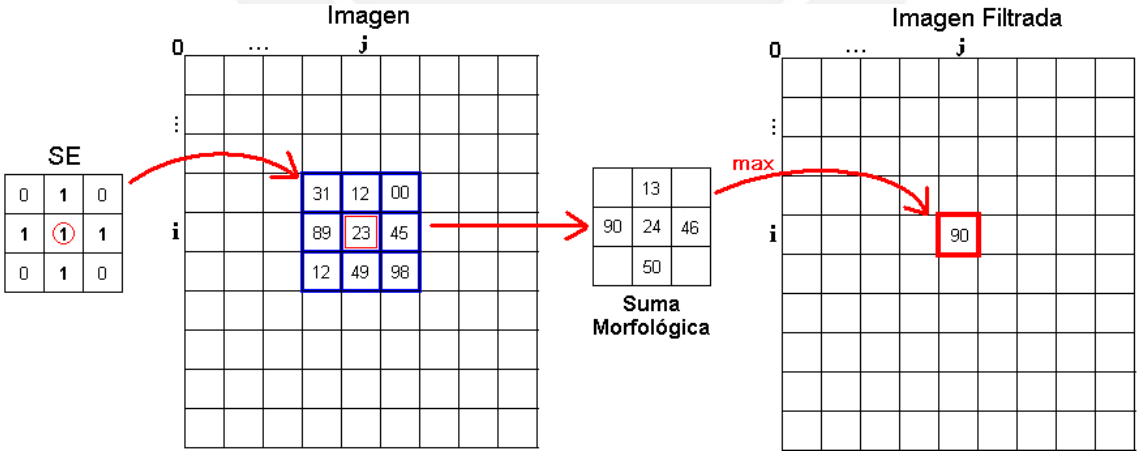


FIGURA 1.2.- CONCEPTO DE FILTRADO DE UNA IMAGEN PARA UN SE DE 3X3 PÍXELES

Los MF se pueden representar en la forma de ecuaciones matriciales, pero cada tipo de filtrado es un algoritmo determinado distinto del resto, sus

operaciones están basadas en el álgebra de operadores no lineales que nacen de la teoría de conjuntos [1]. Aplicar transformaciones morfológicas a una imagen (*A*) con un SE (*B*) significa que B se posiciona sobre cada punto en la imagen y adapta su geometría y magnitud al contexto indicado; en este sentido, se puede decir que *B* cuantifica la descripción de los objetos o estructuras presentes en la imagen *A*.

1.2.- DILACIÓN Y EROSIÓN EN ESCALA DE GRISES

Las operaciones morfológicas en escala de grises se pueden definir a partir de la extensión de las definiciones para el caso de imágenes binarias [1]; en general se definen sobre los espacios Euclidianos n-dimensionales, obteniéndose una serie de operaciones morfológicas en función a las combinaciones de dilaciones y erosiones, que son operaciones duales basadas en la unión e intersección de conjuntos.

La Dilación es la operación morfológica que realiza la unión de conjuntos en una vecindad definida según la condición indicada en la ecuación 1; son todos los puntos para los cuales la intersección del SE con su vecindad no sea nula. Si el elemento estructural tiene una forma simétrica entonces se demuestra que la dilación se puede expresar como una suma de Minkowsky [1] [2], lo cual es mostrado en la ecuación 2.

$$\delta_B A = \{ x \in E^n \ / \ B_x \cap A \neq \phi \} \quad \ldots(1)$$

$$\delta_B A = \bigcup_{b \in B} A_{-b} = A \oplus B \quad \ldots(2)$$

Utilizando el teorema de la descomposición de umbrales [1] se deduce una expresión aritmética para la operación de unión de conjuntos: sea *A* una imagen en escala de grises, donde *A(i,j)* representa el valor del píxel en la fila *i* y la columna *j*; y *B* un elemento estructural, donde el valor del píxel en la fila *m* y la columna *n* está dado por *B(m,n)*. Entonces la dilación se define según la ecuación 3.

$$\delta_B A_{(i,j)} = \max\{\ p\ /\ p = A_{(i-m,j-n)} + B_{(m,n)} \wedge (i-m, j-n) \in DA\ \wedge\ (m,n) \in DB\ \} \quad \ldots(3)$$

Con la vecindad definida por *B* en la imagen original, el resultado de la dilación en el punto *(i,j)* es el máximo valor de la suma de cada píxel, que pertenece al dominio de *A* con el píxel del SE correspondiente. La imagen resultado muestra un aumento de claridad con respecto a la imagen inicial y los elementos oscuros son reducidos o eliminados dependiendo como estén relacionadas las formas al SE.

La Erosión es la operación morfológica que obtiene como resultado una imagen formada por todos los puntos para los cuales se cumple que el SE está contenido en la vecindad que lo define (ver ecuación 4). Se puede interpretar como una intersección de conjuntos según se indica en la ecuación 5, y si el SE es de forma simétrica se puede expresar como una resta de Minkowsky [1] [2].

$$\varepsilon_B A = \{\ x \in E^n\ /\ B_x \subseteq A\ \} \quad \ldots(4)$$

$$\varepsilon_B A = \bigcap_{b \in B} A_{-b} = A \ominus B \quad \ldots(5)$$

De la misma manera, se demuestra que el resultado de la erosión en el punto *(i,j)* es el mínimo valor de la resta de cada píxel, perteneciente tanto a la vecindad definida por *B* como al domino de *A*, con el píxel que le corresponde en *B*. Como consecuencia de esta operación se logra una imagen más oscura con respecto a la imagen original, en la cual fueron reducidas o eliminadas las regiones claras.

$$\varepsilon_B A_{(i,j)} = \min\{\ p\ /\ p = A_{(i-m,j-n)} - B_{(m,n)} \wedge (i-m, j-n) \in DA\ \wedge\ (m,n) \in DB\ \} \quad \ldots(6)$$

Para imágenes de 256 tonalidades de grises, se considera como máximo valor 255 correspondiente al color blanco y como mínimo 0 que corresponde al negro o fondo. Las ecuaciones 3 y 6 operan valores que pertenecen al dominio de los enteros, esto significa que en teoría la dilación de una imagen podría obtener valores numéricos mayores a 255 y la erosión podría dar valores negativos; por ello se implementan operaciones aritméticas de suma y resta como funciones acotadas (ver ejemplo de figura 1.3).

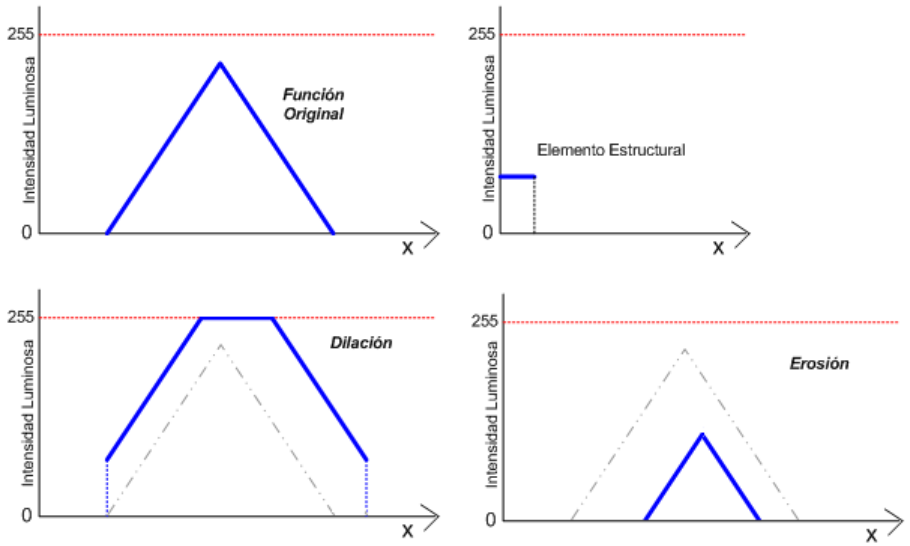


FIGURA 1.3.- DILACIÓN Y EROSIÓN COMO FUNCIONES ACOTADAS

Adicionalmente, el resultado de la suma y resta sobre los píxeles con valor cero obtienen como resultado cero debido a que son parte del fondo de la imagen. Por su parte, un píxel del SE con valor cero indica que el píxel de la imagen enmascarado en aquella posición no es parte de la vecindad y por lo tanto su valor es indiferente al resultado de la operación morfológica.

1.3.- ARQUITECTURAS *PIPELINE*

El análisis de imágenes requiere el manejo y proceso de una gran cantidad de datos a altas velocidades de procesamiento, por ello surge la importancia de procesar la información en paralelo. Algunas décadas atrás se empezaron a diseñar procesadores basados en arquitecturas síncronas para procesar datos en paralelo, algunos ejemplos son los procesadores vectoriales, *SIMD* y de arreglos sistólicos [6]; una arquitectura *pipeline* es un tipo de arreglo sistólico [13]. Los procesadores basados en arquitecturas *pipeline* son comúnmente empleados para realizar operaciones sobre vecindades en imágenes de manera eficiente, tales como convolución, correlación y operaciones morfológicas [6] [14].

Una arquitectura *pipeline* contiene un arreglo de etapas de procesamiento [7], el cual consiste en una cadena de registros de almacenamiento cuyos datos son desplazados continuamente en una sola dirección (ver figura 1.4). La información de cada registro es parte de una etapa y se deriva a un bloque de procesamiento que puede realizar una operación particular, luego esta información es entregada al registro de la siguiente etapa junto con un

resultado parcial. Una ventaja de esta arquitectura radica en el hecho que la información de la imagen no necesita ser almacenada en arreglos complicados, solo se requieren simples interconexiones; además, su funcionamiento solo requiere de señales de sincronización que son entregadas por una unidad de control.

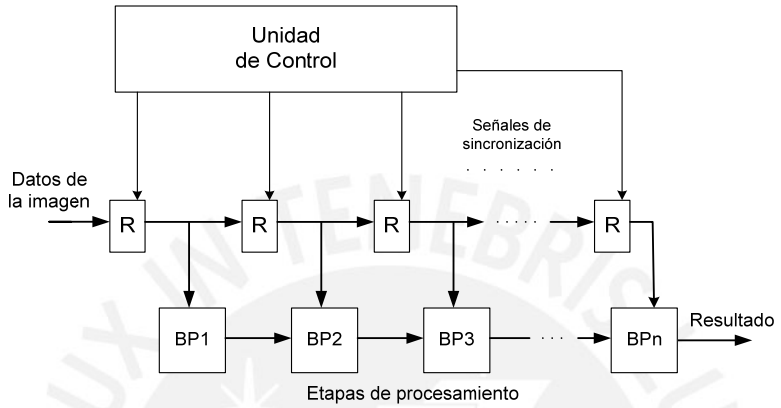


FIGURA 1.4.- CADENA DE REGISTROS EN UNA ARQUITECTURA *PIPELINE*

La entrada de los datos se realiza de manera secuencial, por esta razón los píxeles de una imagen se organizan en una matriz unidimensional (ver figura 1.5). Los datos entrantes se mueven a través de los registros, y solo se obtienen los resultados a partir del momento en que el primer dato entrante ha llegado al registro de la última etapa. Posteriormente, la salida de resultados se realiza en simultáneo con la entrada, lo que constituye un flujo continuo.

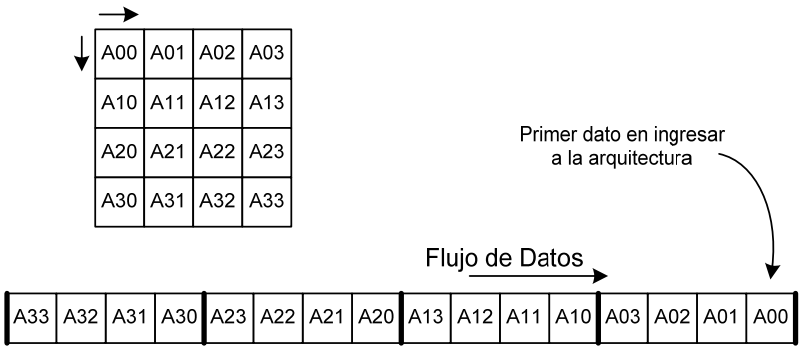


FIGURA 1.5.- MATRIZ UNIDIMENSIONAL PARA UNA IMAGEN DE 4X4 PÍXELES

La cadena de registros provee la información de un grupo consecutivo de píxeles de una imagen, esto se asemeja a realizar el enmascaramiento de aquella región de puntos. A medida que se realiza el flujo de datos, distintas regiones de la imagen logran ser enmascaradas, lo que permite la implementación de operaciones de filtrado morfológico, de esta manera se realizan las operaciones lógicas y numéricas respectivas con los datos de la vecindad almacenados temporalmente.

Las operaciones morfológicas en escala de grises implican una interacción directa entre los píxeles de la imagen y el SE, según las ecuaciones 3 y 6 (ver sección 1.2). Por ello se requiere implementar la suma y resta de los píxeles del elemento estructural en los puntos definidos como vecindad. Entonces el número de unidades aritméticas requeridas es el mismo número de píxeles del SE, tanto para la dilación y erosión, asimismo se asume que todos los píxeles del SE están siempre disponibles para cada etapa del *pipeline* almacenados en registros separados (ver figura 1.6).

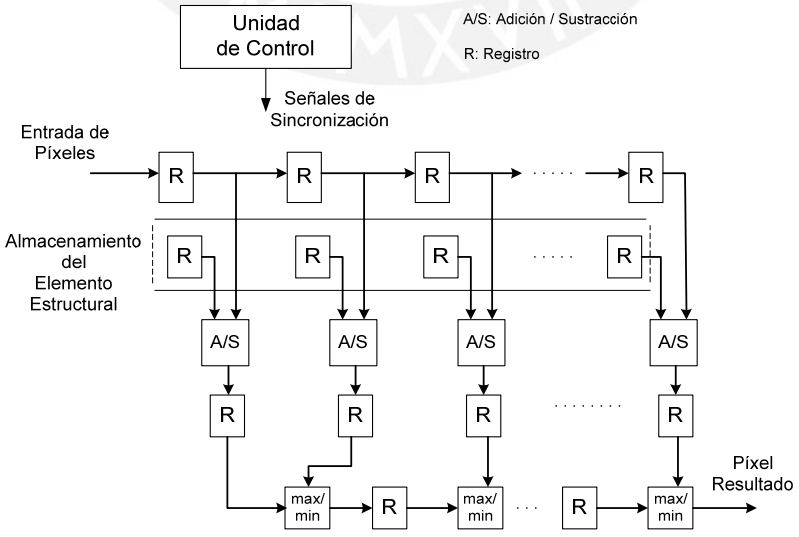


FIGURA 1.6.- ARQUITECTURA *PIPELINE* PARA LA IMPLEMENTACIÓN DE LAS OPERACIONES DE DILACIÓN Y EROSIÓN

Es posible implementar circuitos basados en *pipeline* que realicen filtrado morfológico en una dimensión y luego extender el proceso a dos dimensiones mediante la agrupación de bloques de procesamiento unidimensional en paralelo, uno para cada fila diferente. Finalmente los resultados de cada bloque unidimensional son comparados obteniéndose el valor máximo y mínimo, correspondientes a los resultados de dilación y erosión respectivamente. Filtrar una imagen con un elemento estructural de tamaño 3x3 píxeles requiere los datos de una vecindad de 9 píxeles pertenecientes a la imagen enmascarada; se necesita la información de 3 filas consecutivas en paralelo, donde cada una de ellas se procesa con un elemento estructural de tamaño 3x1 píxeles (ver figura 1.7).

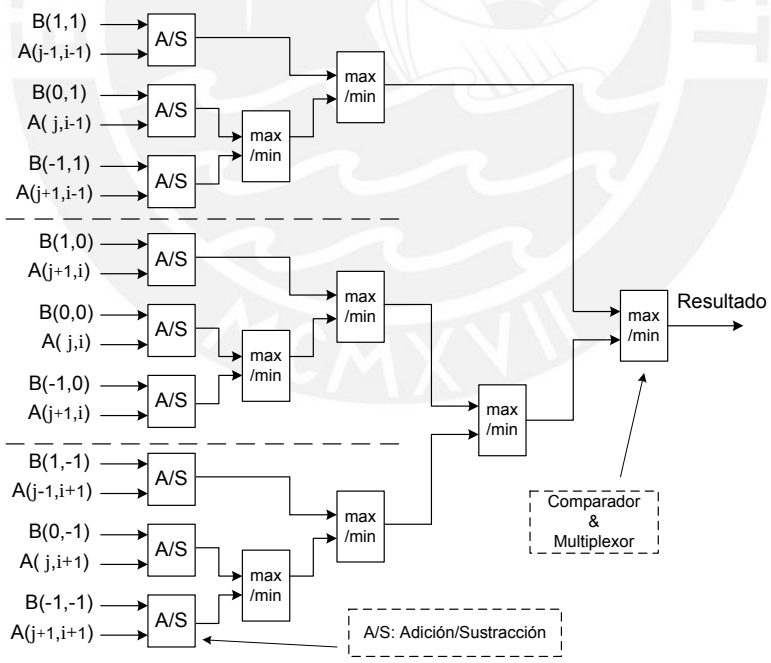


FIGURA 1.7.- ESQUEMA DE LOS BLOQUES ARITMÉTICOS PARA LAS OPERACIONES DE DILACIÓN Y EROSIÓN SOBRE UNA VECINDAD DE 3X3 PÍXELES

La unidad de control provee además señales que indican cuando el centro del SE está posicionado sobre alguno de los cuatro bordes de la imagen: superior, inferior, izquierdo y derecho (ver figura 1.8). Para esto se asume que la imagen esta rodeada por píxeles nulos.

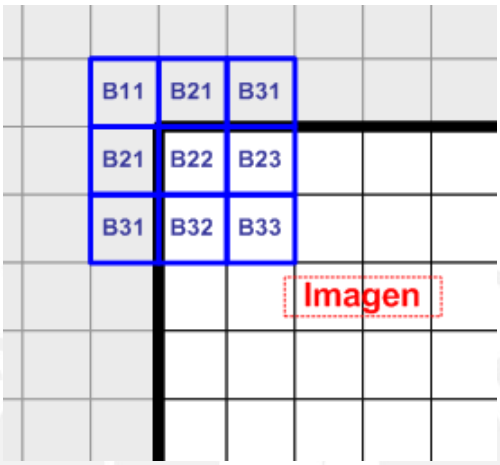


FIGURA 1.8.- ELEMENTO ESTRUCTURAL POSICIONADO EN LA ESQUINA SUPERIOR IZQUIERDA DE UNA IMAGEN

2.- DISEÑO DE LA ARQUITECTURA

2.1.- INTRODUCCIÓN

La arquitectura diseñada configura a un FPGA [5] como un filtro morfológico de imágenes en escala de grises que realiza las operaciones de dilación y erosión de manera simultánea utilizando un SE con un tamaño de 3x3 píxeles. Para ello se utiliza un esquema *pipeline* [7] (ver figura 2.1) que incorpora un bloque de almacenamiento temporal, en donde se guardan las filas de la imagen mientras son ingresadas al sistema, previo a las etapas de filtrado unidimensional.

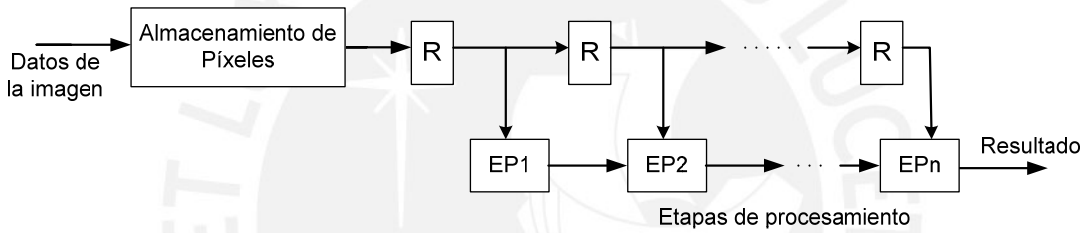


FIGURA 2.1.- ESQUEMA *PIPELINE* BASADO EN ETAPAS DE PROCESAMIENTO

La unidad de control diseñada hace que la entrada de datos se realice de manera secuencial, inicialmente la información de los píxeles del SE es ingresada y posteriormente la información de la imagen; los píxeles del SE están disponibles como operadores en los bloques aritméticos durante el proceso de una imagen y por ello son almacenados previamente, sus valores no son alterados durante el proceso. En cambio los datos de la imagen se desplazan a través de las distintas etapas de la arquitectura siendo actualizados con los píxeles no procesados de la imagen. Con ello, en cada etapa de procesamiento los datos se operan secuencialmente, y una vez que

su resultado es obtenido este se transfiere a la siguiente etapa de procesamiento.

Toda la arquitectura fue implementada con el lenguaje de descripción de hardware VHDL [15] [16], lo que permite trasladar este diseño a distintos modelos de FPGAs. Además se utilizaron las herramientas Max+Plus II y Quartus II [17] de Altera para diseño digital en la compilación y simulación de la arquitectura y de cada módulo que la conforma. Los diferentes módulos poseen cada uno su propio código VHDL independiente; estos códigos fueron escritos de manera parametrizada, lo que significa que las constantes que indican el número de bits del bus de datos, ancho de palabra de contadores y tamaño de memoria, dependan de variables genéricas que se definen al inicio de cada código, y hacen al diseño flexible a cambios de parámetros.

Se pueden procesar imágenes de tamaño variable, teniendo como límite un valor potencia de dos que es fijado por dos variables genéricas (ancho y alto de la imagen); a diferencia de otros trabajos que solo pueden procesar imágenes de un tamaño fijo, una vez que se han implementado. Esto se logra debido a que la unidad de control trabaja en base a contadores, los cuales indican las coordenadas de fila y columna del píxel en proceso, cuyos valores de cuenta máxima se configuran al inicio de cada proceso según el tamaño de la imagen. La implementación de la arquitectura presentada se configuró para procesar imágenes en escala de grises de 256 tonos, teniendo un tamaño máximo de prueba de 256x256 píxeles.

2.2.- DESCRIPCIÓN GENERAL

El diseño implementado se divide en etapas a través de las cuales se realiza un flujo de datos (ver figura 2.2); estas etapas son: registros de almacenamiento del SE, memoria de almacenamiento temporal de filas de la imagen, ordenador de filas almacenadas, 3 bloques en paralelo de filtrado morfológico unidimensional, y un comparador de resultados parciales que obtiene el máximo y mínimo valor procesado.

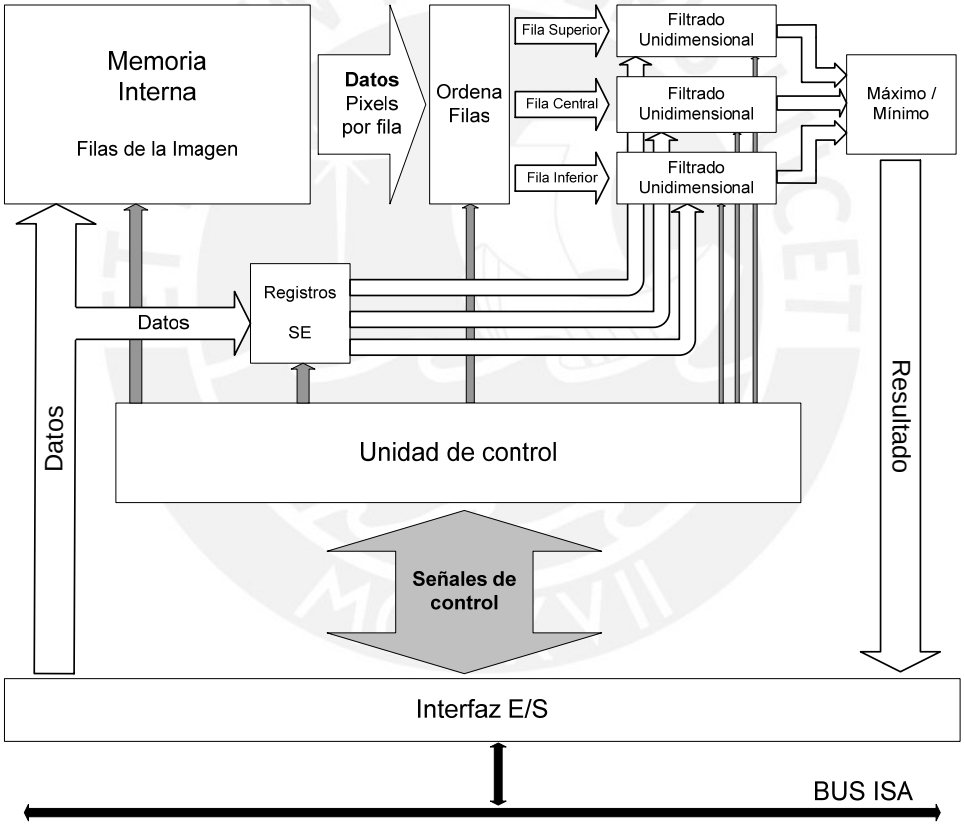


FIGURA 2.2.- DIAGRAMA DE BLOQUES DE LA ARQUITECTURA

El bloque de registros de almacenamiento del SE agrupa su información en 3 filas, y cada una es asignada a un bloque de filtrado unidimensional según su orden, este orden se mantiene durante todo el proceso de la imagen. Por su parte, la memoria almacena temporalmente la información de 3 filas

consecutivas de la imagen original, las cuales son indispensables en la obtención de una fila de la imagen resultado; además en el filtrado de una nueva fila, la información contenida se renueva con la información de una nueva fila, siendo reemplazada la fila más antigua. Cada bloque de almacenamiento posee señales de habilitación de escritura y direccionamiento provenientes de la unidad de control. La descripción en VHDL de la unidad de control junto con los bloques de almacenamiento se encuentra en *CONTROL_DATOS.VHD* (ver anexo A.1).

Seguidamente al bloque de memoria, el bloque *Ordena_Filas* se encarga de asignar consecutivamente la información de las filas almacenadas hacia los bloques de filtrado unidimensional, asignando la fila más antigua como la fila superior mientras que a la recientemente ingresada la fila inferior. La descripción en VHDL de este bloque se encuentra en *ORDENA_FILAS.VHD* (ver anexo A.2).

Cada bloque de filtrado unidimensional funciona de acuerdo al esquema *pipeline* (ver figura 2.1) para vecindades de tamaño 3x1; el código en VHDL se presenta en *FILTRADO_UNIDIM.VHD* (ver anexo A.3), donde se describe una cadena de registros y se incorpora modularmente bloques aritméticos a la salida de sus registros. Sus resultados pasan a un bloque que obtiene el máximo y mínimo valor, que son los resultados finales.

En la unidad de control se administra la transferencia de datos con una interfaz E/S al Bus ISA mediante el uso de banderas de control. Adicionalmente, la

sincronización de las etapas mostradas y el direccionamiento de la memoria y los registros del SE, son gobernadas en base a seis contadores de control:

- Contador SE (*C_SE*): Realiza el direccionamiento en los registros del SE durante el ingreso de los píxeles que lo conforman.
- Contador de direccionamiento de fila en la memoria (*im*): Realiza el direccionamiento de filas de la imagen en el bloque de memoria.
- Contador de columnas de imagen entrante (*j*)
- Contador de filas de imagen entrante (*i*)
- Contador de columnas de píxel procesado (*opj*)
- Contador de filas de píxel procesado (*opi*)

Las coordenadas de la columna y la fila de los píxeles de la imagen que ingresan a memoria están dadas por los contadores *j* e *i* respectivamente, además *j* realiza el direccionamiento de la columna en cada fila almacenada en la memoria. Los contadores *opj* y *opi* indican las coordenadas de columna y fila respectivamente del píxel de la imagen resultado; además, sus cuentas están desfasadas en tiempo con respecto a *j* e *i* (ver figura 2.3).

FIGURA 2.3.- SECUENCIA DE LOS CONTADORES DURANTE EL PROCESO DE UNA IMAGEN DE 8X8 PÍXELES

El desfase se produce porque los contadores *opj* y *opi* inician su cuenta desde que se tiene almacenada la información suficiente para obtener la primera fila de la imagen resultado, entonces se inicia un flujo continuo de entrada y salida de datos. El ingreso de datos de la imagen a la memoria termina cuando *j* e *i* llegan a sus máximos valores de cuenta. Después se termina el proceso de una imagen, cuando *opj* y *opi* llegan al máximo valor de cuenta, entonces todos los contadores son detenidos por la unidad de control.

Todos los bloques previamente descritos se integran dentro del módulo *MORFOVAR.VHD* (ver anexo A.5) el cual no contiene al bloque de la interfaz E/S, lo que lo hace portátil a implementaciones con diferentes dispositivos (*hardware)*. El bloque *MM_GRIS.VHD* incluye adicionalmente la interfaz E/S al Bus ISA (ver anexo A.6.2), con el cual se llevaron a cabo las pruebas experimentales del diseño.

2.3.- ETAPAS DE PROCESAMIENTO

2.3.1.- GENERACIÓN DE ESCRITURA EN LOS REGISTROS DE ALMACENAMIENTO DEL SE

Los píxeles del SE son almacenados de manera matricial en 9 registros, siendo estos registros agrupados en 3 filas de 3 columnas (ver figura 2.4). Todos ellos comparten el mismo bus de entrada de datos, pero tienen diferentes habilitadores de escritura.

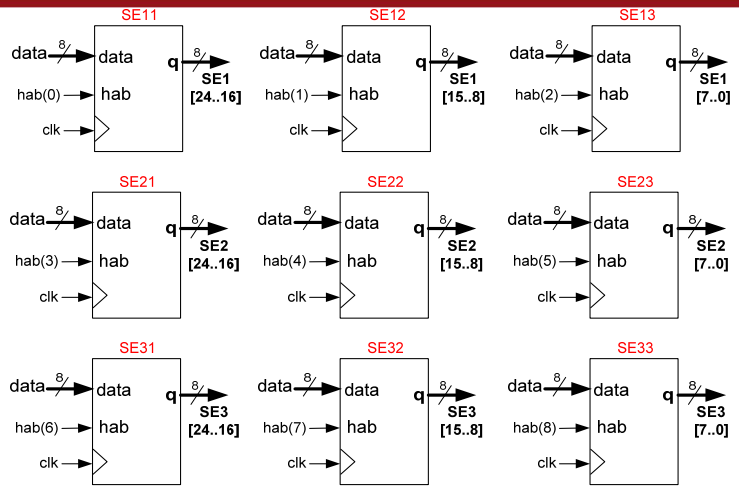


FIGURA 2.4.- REGISTROS DE ALMACENAMIENTO DEL SE

La palabra *habil_SE* (ver figura 2.5) habilita la escritura en los registros del SE, pero ésta depende del valor del contador *C_SE* y de una señal proveniente de la unidad de control (*Escribe_SE*); por ello el bloque de generación de escritura es un decodificador del contador *C_SE*, que es habilitado por la señal *Escribe_SE*. Entonces, durante el proceso de escritura del SE solo se habilita un registro a la vez, y luego de completado el proceso, los registros no se alteran hasta que se reinicie el sistema.

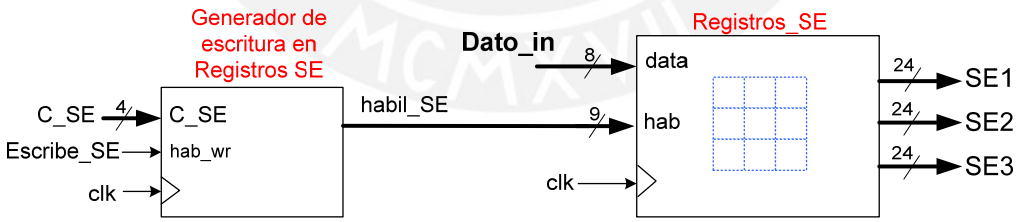


FIGURA 2.5.- HABILITACIÓN DE LA ESCRITURA EN REGISTROS DEL SE

2.3.2.- GENERACIÓN DE ESCRITURA EN EL BLOQUE DE MEMORIA

Durante el proceso de una imagen se dispone de la información de tres filas consecutivas almacenadas en memoria, de tal manera que sean procesadas

en paralelo, cada una con la fila del SE correspondiente. Para lograr esto se ingresa a la memoria la información de la fila subsiguiente mientras se opera con las filas previas y con ello operar la fila recientemente ingresada, inmediatamente al término del procesamiento de la anterior, junto con las dos previamente almacenadas y menos antiguas.

Por estas razones, se divide el bloque de memoria en 4 unidades de memoria, siendo cada una destinada a almacenar temporalmente una fila de la imagen (ver figura 2.6). Cada unidad se denota como LPM_RAM_DQ [17], y funciona de tal manera que siempre se lee en su salida *q[ ]* el valor del byte almacenado en una determinada dirección de memoria, la cual se indica con la entrada *address*. La escritura de un dato se realiza cuando la señal *we* se pone en alta, entonces el dato colocado en la entrada *data[ ]* se deposita en la dirección de memoria seleccionada.

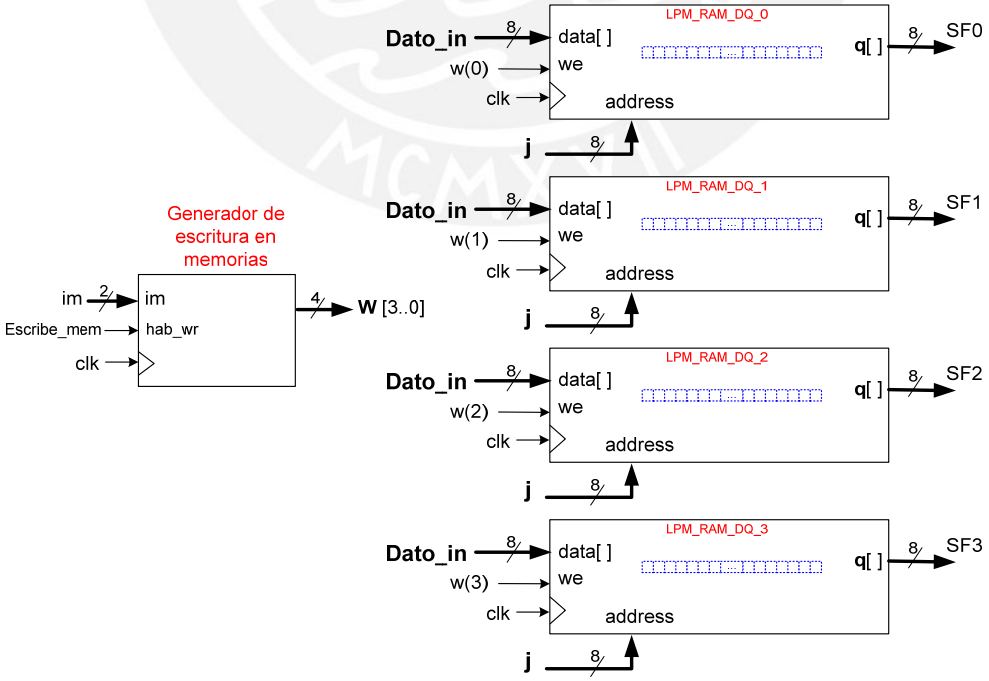


FIGURA 2.6.- GENERADOR DE ESCRITURA EN LA RAM INTERNA

El valor del contador *j* realiza el direccionamiento en las 4 unidades de memoria; dado que cada una almacena la información de una fila de la imagen, con lo que el contador *j* selecciona la columna dentro de cada fila. Por su parte, el valor del contador *im* indica en que unidad de memoria escribir.

El bloque generador de escritura funciona como decodificador de *im*, y su habilitación depende de la señal de control *Escribe_mem* proveniente de la unidad de control. Solo se escribe en una unidad de memoria a la vez, en la dirección indicada por *j*; mientras que las otras unidades solo indican su contenido respectivo en la misma dirección indicada, adicionalmente este contenido se transfiere al bloque *ordena_filas* y posteriormente a los bloques de filtrado unidimensional para su procesamiento.

A medida que el valor de *j* se incrementa desde cero hasta el máximo valor de cuenta se van cubriendo todas las posiciones dentro de cada unidad de memoria. Cuando *j* llega al máximo valor de cuenta, la unidad de control lo reinicia e incrementa el valor de *im*, logrando que se empiece a escribir desde el principio en la unidad de memoria siguiente. Luego que *im* selecciona la última unidad de memoria su valor vuelve a cero, lo cual indica la escritura en la primera unidad de memoria; este proceso se repite cíclicamente hasta finalizar el proceso de una imagen. De esta manera el ingreso de una nueva fila siempre reemplaza el contenido de la más antigua.

2.3.3.- ORDENAMIENTO DE FILAS ALMACENADAS

Las filas de la imagen almacenadas en memoria son aquellas sobre las que se encuentra posicionado un SE de 3x3 píxeles, el cual está centrado en la columna *opj*; el píxel de la fila subsiguiente que está en proceso de almacenamiento se encuentra en la columna *j* (ver figura 2.7). Cuando el valor de *opj* se incrementa, esto se traduce a un desplazamiento del SE hacia la derecha; de la misma manera ocurre con la fila subsiguiente, la selección del píxel que se actualiza en memoria también se va desplazando una posición hacia la derecha.

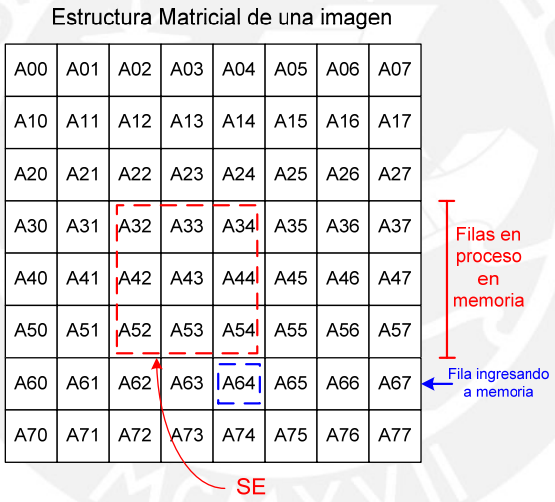


FIGURA 2.7.- EJEMPLO DEL POSICIONAMIENTO DE UN SE DE 3X3 PÍXELES SOBRE UNA IMAGEN DE 8X8 PÍXELES

Según la arquitectura del bloque de memoria, las filas no siempre se almacenan en el orden consecutivo de arriba hacia abajo (ver figura 2.8). Por ello se dispone de un circuito adicional que ordene las filas, manteniendo siempre a la más antigua como fila superior y a la recientemente almacenada como la inferior. De esta manera, la información respectiva es asignada a los bloques de filtrado unidimensional.

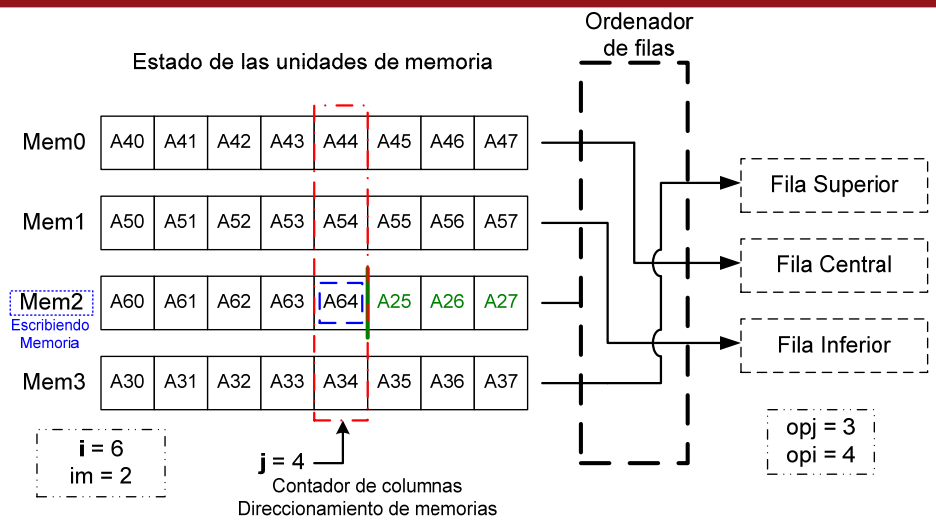


FIGURA 2.8.- FUNCIONAMIENTO DEL ORDENADOR DE FILAS. EJEMPLO PARA EL PROCESO DE UNA IMAGEN DE 8X8 PÍXELES

En el bloque *Ordena_filas*, la información proveniente de las 4 unidades de memoria (M0, M1, M2, M3) es agrupada en 3 salidas (F0, F1, F2) por un conjunto de tres multiplexores, según el orden establecido (ver figura 2.9); nunca se asigna como salida la información de la unidad de memoria en la cual se está escribiendo.

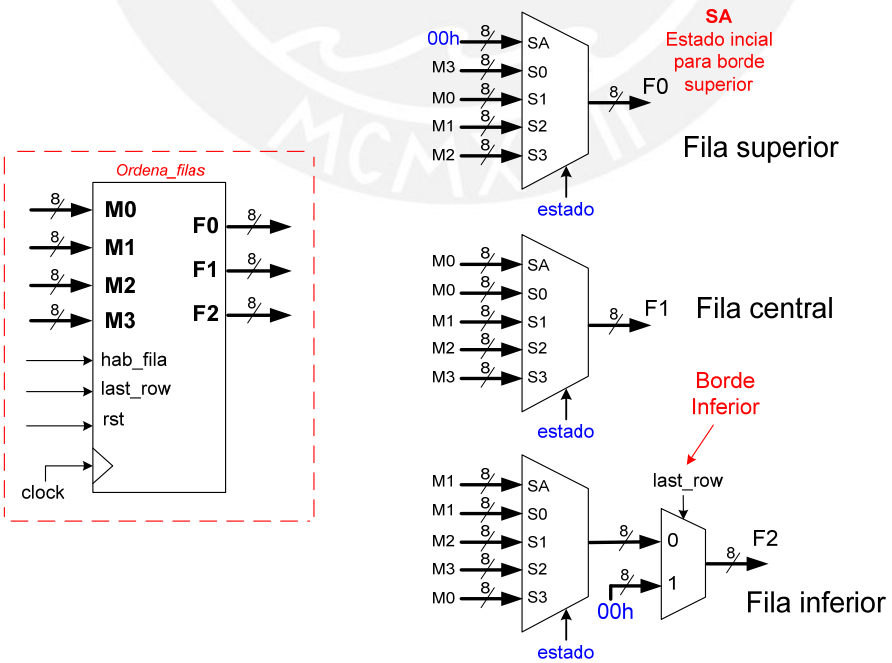


FIGURA 2.9.- DIAGRAMA LÓGICO DEL BLOQUE ORDENA_FILAS

Por otro lado, este diseño tiene la ventaja de facilitar siempre la información de tres filas en paralelo, pero se generan anomalías cuando se procesan los bordes superior e inferior de la imagen resultado. Al procesar los datos para obtener la primera fila de la imagen resultado se necesita solo la información de las dos primeras filas de la imagen original que están almacenadas en dos unidades de memoria, pero también se brinda la información de una tercera unidad de memoria, cuya información también sería procesada, obteniendo un resultado final erróneo; una situación similar se genera en el momento que se procesa la última fila de la imagen resultado (ver figura 2.10).

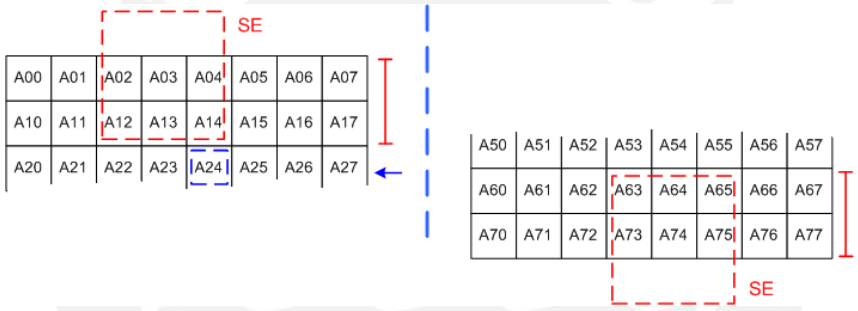


FIGURA 2.10.- POSICIONAMIENTO DEL SE SOBRE LOS BORDES SUPERIOR E INFERIOR EN UNA IMAGEN DE 8X8 PÍXELES

La asignación de los multiplexores está gobernada por una máquina de cinco estados (SA, S0, S1, S2 y S3). La operación de la primera fila de la imagen resultado se da siempre en el estado SA, cuya asignación es similar al estado S0, pero se diferencia en que se asigna un valor nulo a la fila *F0*, así se elimina el efecto de borde superior en la imagen. Por otro lado, cuando la señal *last_row* se pone en alta, la unidad de control indica el procesamiento de la última fila de la imagen resultado, entonces se consigue que la asignación de la fila inferior *F2* sea nula; eliminándose el efecto de borde inferior.

Los cambios de estado (ver figura 2.11) dependen de la señal de control *hab_fila*, que está relacionada al incremento del contador *im*. Por su parte, la activación de la señal de control asíncrona *rst* permite el retorno al estado inicial SA desde cualquier estado.

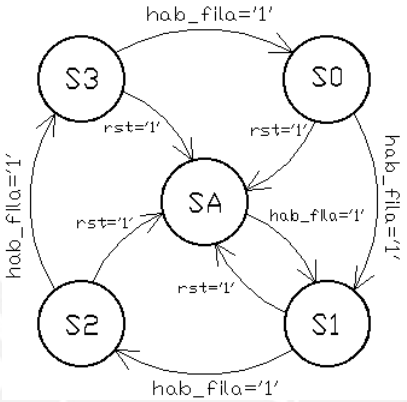


FIGURA 2.11.- DIAGRAMA DE ESTADOS DEL BLOQUE *ORDENA_FILAS*

La forma de ordenamiento del bloque *Ordena_filas* junto con el bloque de memoria, logran un *pipeline* [7] con las filas de la imagen, siendo estas desplazadas en dirección ascendente (ver figura 2.12).

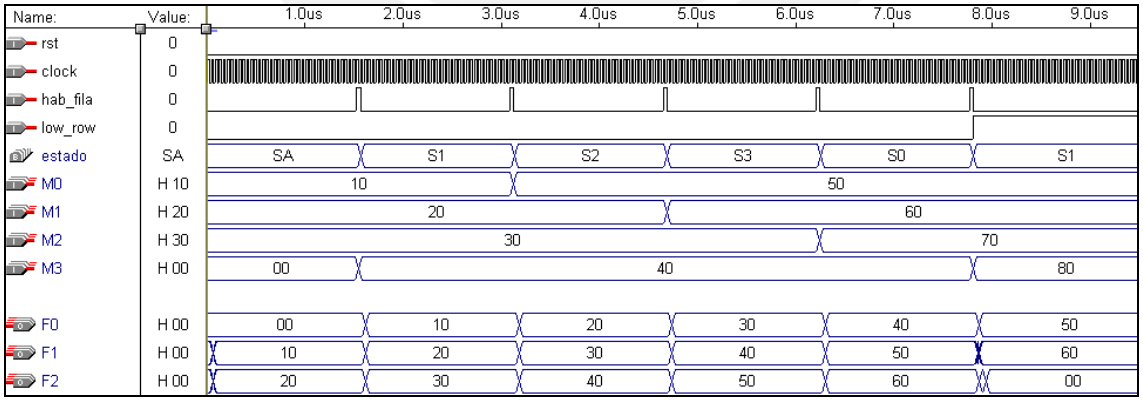


FIGURA 2.12.- SIMULACIÓN DEL BLOQUE *ORDENA_FILAS*

2.3.4.- FILTRADO UNIDIMENSIONAL

Se cuentan con 3 bloques que realizan el filtrado morfológico a las filas almacenadas en la memoria, a cada uno se le asigna permanentemente una fila del SE, el valor de un píxel perteneciente a una fila almacenada y sus respectivas señales de sincronización (ver figura 2.13). Con lo que se obtienen los resultados unidimensionales para cada fila de las operaciones de dilación y erosión en paralelo.

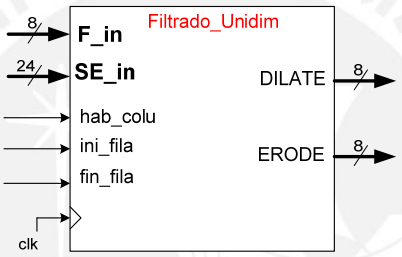


FIGURA 2.13.- SEÑALES DE UN BLOQUE DE FILTRADO UNIDIMENSIONAL

La información de cada fila de la imagen es transferida como una trama continua de datos (ver figura 2.14). Cada vez que un SE de tamaño 3x1 píxeles se desplaza una posición a la derecha, consigue enmascarar un píxel nuevo, dejando de lado al píxel que se encontraba a la izquierda, pero continúa enmascarando a los otros dos píxeles que había enmascarado en la posición previa.

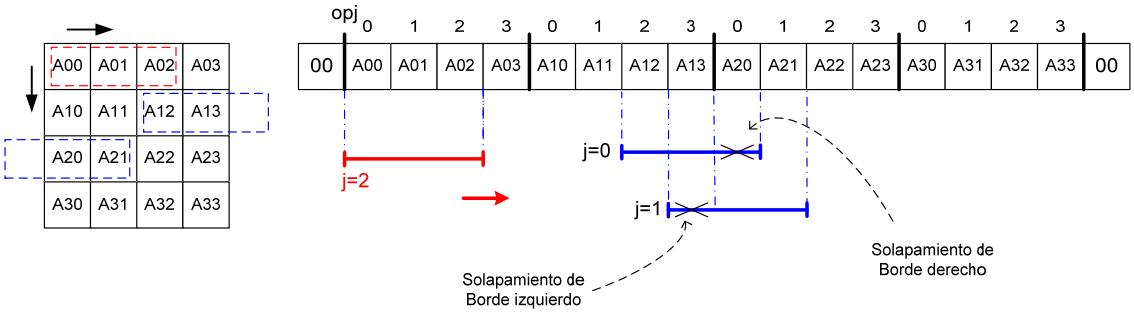


FIGURA 2.14.- EJEMPLO DEL ENMASCARAMIENTO DE UN SE DE 3X1 PÍXELES A TRAVÉS DE UNA IMAGEN DE 4X4 PÍXELES

La trama de datos se introduce a una cadena de registros que almacena temporalmente los datos que son asignados a un píxel del SE (ver figura 2.15). La información ingresada (*F_in*) es la del píxel correspondiente a la columna *j*, el cual proviene de la memoria. Con cada incremento del contador *j*, la señal de control *hab_colu* se pone en alta por un instante, permitiendo que el contenido del dato ingresado se guarde en un primer registro (*Sb*), el contenido previo de este es transferido a un segundo registro (*Sc*), y a su vez la información de entrada se actualiza con el contenido de la posición de memoria siguiente.

Este planteamiento genera enmascaramientos no deseados cada vez que el centro del SE es colocado sobre el borde izquierdo o derecho de una fila:

- Cuando se procesa el primer píxel de una fila, también se enmascara el último píxel de la fila anterior, lo que origina un efecto de borde izquierdo.
- Al momento de procesar el píxel correspondiente a la última columna de una fila, también se enmascara el primer píxel de la fila siguiente, lo cual genera un solapamiento o efecto de borde derecho.

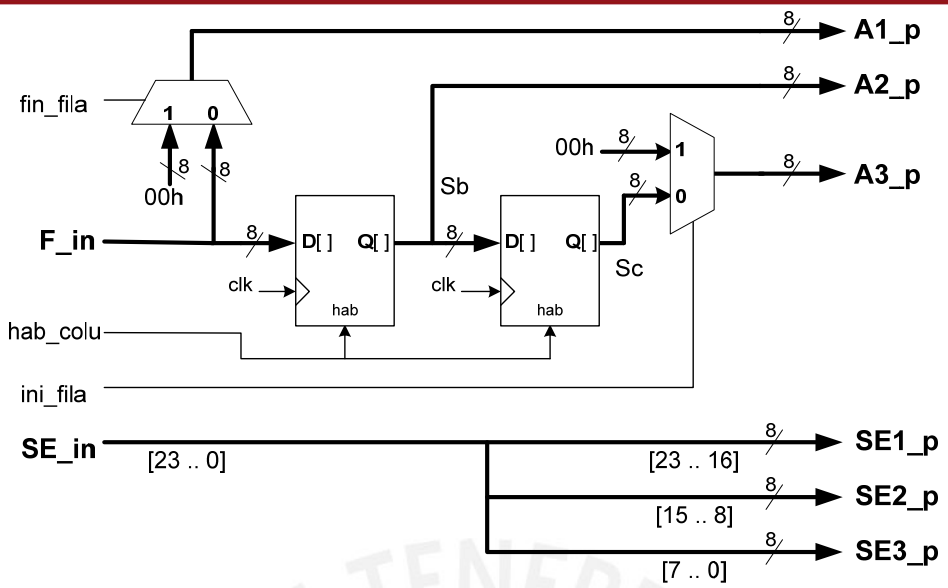


FIGURA 2.15.- DIAGRAMA LÓGICO DEL BLOQUE *FILTRADO_UNIDIM*

El registro *A3_p* contiene el valor del píxel enmascarado a la izquierda, *A2_p* contiene el valor del píxel central y *A1_p* contiene el valor del píxel enmascarado a la derecha. Las señales de control *ini_fila* y *fin_fila* se encargan de anular los efectos de borde mediante la asignación de cero sobre los registros *A1_p* y *A3_p*, usando los multiplexores respectivos.

Se agrupa cada registro (*A1_p*, *A2_p*, *A3_p*) con un píxel correspondiente del elemento estructural (*SE1_p*, *SE2_p*, *SE3_p*) que son ingresados a tres sumadores y tres restadores morfológicos. Los resultados de las sumas morfológicas entran a un bloque que obtiene la máxima suma, consiguiéndose el resultado de la dilación; mientras que resultados de las restas morfológicas entran a un bloque que obtiene la mínima resta, consiguiéndose la erosión. Ambas operaciones se dan a través de bloques aritméticos en paralelo (ver figura 2.16).

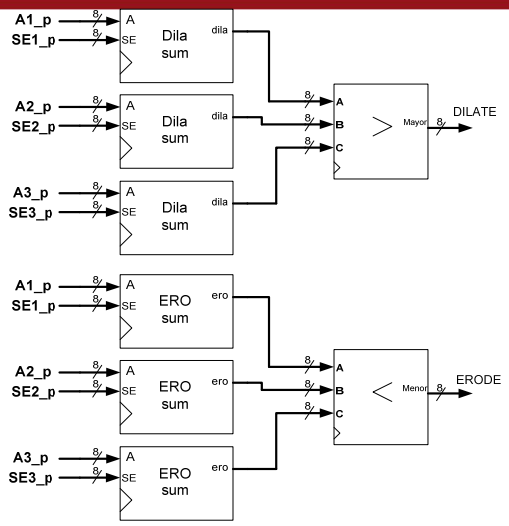


FIGURA 2.16.- FILTRADO MORFOLÓGICO EN PARALELO PARA LOS RESULTADOS DE UNA FILA

2.3.5.- COMPARACIÓN DE RESULTADOS DE LOS FILTROS UNIDIMENSIONALES

Con los resultados de las operaciones de dilación y erosión de los tres bloques de filtrado unidimensional, se logra el resultado final de un píxel mediante su comparación, obteniéndose el máximo y el mínimo valor para los resultados de dilación y erosión respectivamente (ver figura 2.17). Los módulos de comparación y obtención de máximo/mínimo son los mismos que se utilizan en la última etapa de filtrado unidimensional. Los resultados de dilación y erosión son transferidos en paralelo a la interfaz E/S.

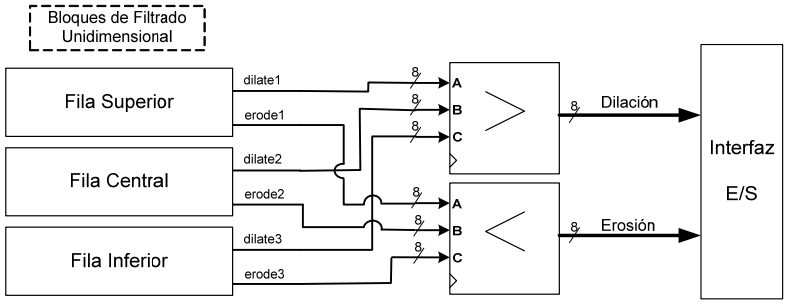


FIGURA 2.17.- ETAPA DE COMPARACIÓN DE RESULTADOS UNIDIMENSIONALES

2.4.- BLOQUE DE CONTROL

La administración del flujo secuencial de datos a través de las distintas etapas de la arquitectura se da en el bloque de control, en el cual se puede distinguir la interacción de dos unidades, que son la unidad central de control y otra de contadores, que juntas gobiernan la secuencia del proceso de una imagen.

La unidad central de control y el bloque de contadores se comunican en todo momento, poniéndose de acuerdo en el envío de las señales de sincronización y de control (ver figura 2.18), como la escritura en memoria y los Registros del SE, además de manejar la interfaz E/S.

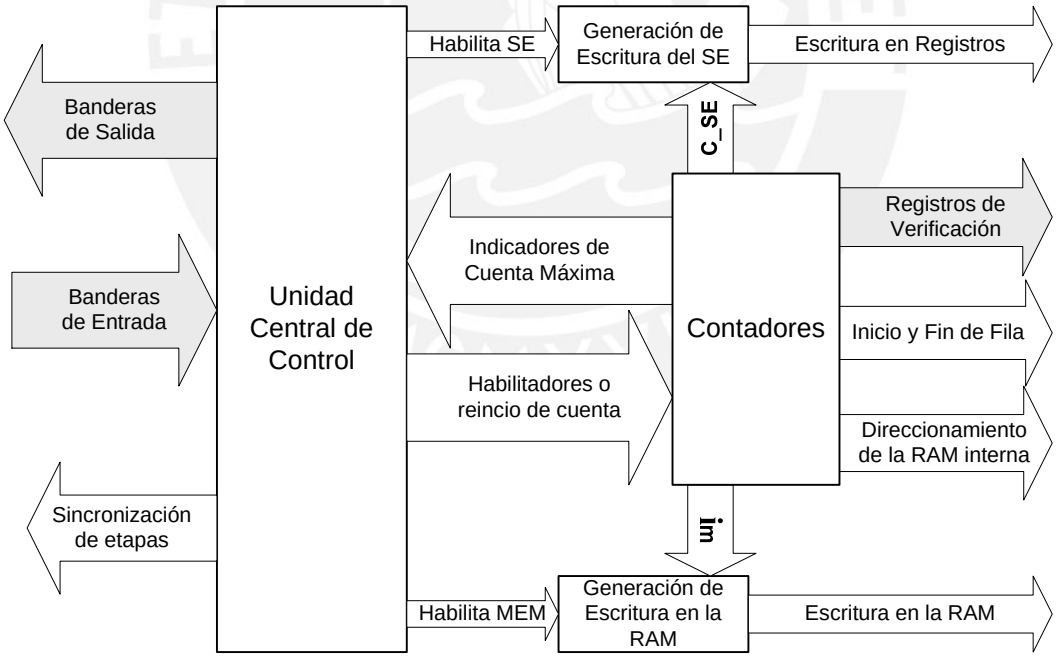


FIGURA 2.18.- ESQUEMA FUNCIONAL DEL BLOQUE DE CONTROL

La unidad central de control funciona en base a una máquina de estados, por ello las señales de sincronización que administra (ver figura 2.19) se dan de acuerdo al estado en que se encuentre.

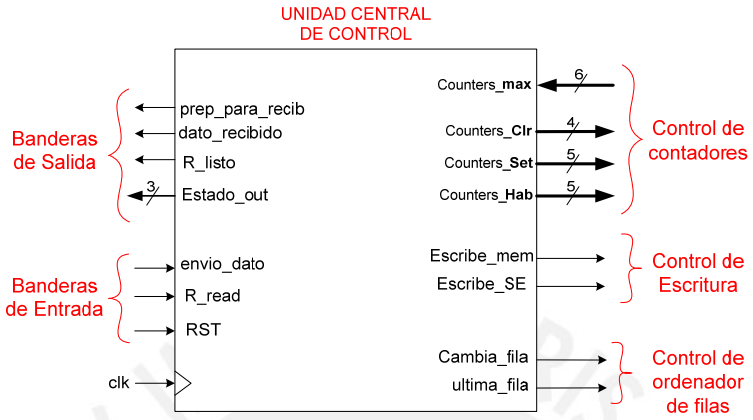


FIGURA 2.19.- SEÑALES DE LA UNIDAD CENTRAL DE CONTROL

Las tareas que realiza la unidad central de control son las siguientes:

- Controla la secuencia de los contadores.
- Asigna señales de sincronización a las etapas del sistema.
- Habilita la escritura en el bloque de memoria y los Registros del SE.
- Controla la transferencia de datos de la Interfaz E/S usando banderas de control.

2.4.1.- CONTROL DE CONTADORES

Tiene la función de indicarle la secuencia del proceso de una imagen a la unidad central. Este bloque consta de seis contadores:

- Contador de direccionamiento de registros SE (*C_SE*)
- Contador de direccionamiento de fila en la RAM (*im*)
- Contador de columnas de imagen entrante (*j*)
- Contador de filas de imagen entrante (*i*)

- Contador de columnas de píxel procesado (*opj*)
- Contador de filas de píxel procesado (*opi*)

El bloque de contadores es administrado en base a señales de habilitación de cuenta (*hab*), señales de reinicio asíncrono de cuenta (*clr*) y señales de fijación asíncrona que sirve para cargar el valor de la cuenta máxima (*set*), todas ellas son asignadas por la unidad central de control (ver figura 2.20). Además cada contador posee una línea que es etiquetada con sufijo *max*, la cual se pone en alta en el momento que su valor de cuenta iguala el contenido de un registro respectivo a cada uno que indica su cuenta máxima.

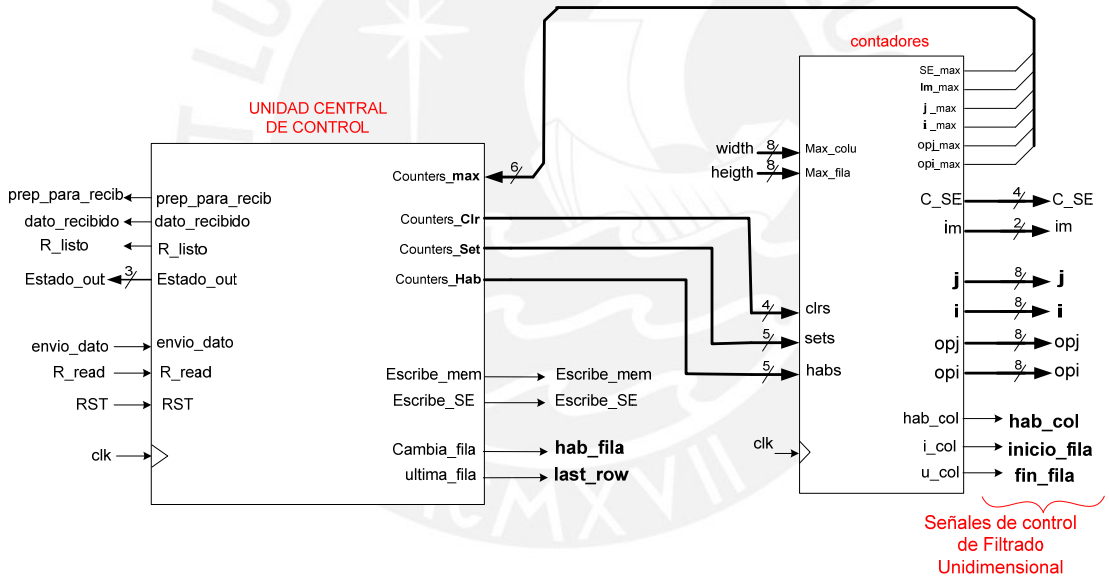


FIGURA 2.20.- SEÑALES DE CONTROL DEL BLOQUE DE CONTADORES

El contador *C_SE* pone en alta su línea *max* cuando su cuenta llega a igualar el número de píxeles del SE (9), lo que indica que todos los registros del SE fueron almacenados, en ese momento su cuenta se detiene. Por su parte el contador *im* pone en alta su línea *max* al igualar el valor de 2, que es el momento en el que hay suficientes datos en memoria para empezar a procesar

la imagen. Ambas líneas *max* solo tienen relevancia en dos instantes a lo largo de todo el proceso, que es cuando son utilizadas para indicar dos cambios de estado (Ingreso del SE y Almacenamiento de las 2 primeras filas de la imagen).

Los valores de cuenta máxima para los contadores de columnas (*j*, *opj*) y filas (*i*, *opi*) están dados por los registros de entrada de ancho de imagen (*width*) y de altura de imagen (*heigth*) respectivamente, la configuración de estos registros se da antes del inicio del proceso de cada imagen , dando facilidades al usuario para procesar imágenes de tamaño variable. En el momento en que alguno de los contadores de columnas alcanza el valor contenido en el registro *width*, la señal *max* respectiva (*j_max* u *opj_max*) se activa y la unidad de control reinicia la cuenta al poner en alta por un instante la línea de control *clear*, a su vez se incrementa la cuenta del contador de fila correspondiente. De manera similar, cuando alguno de los contadores de filas iguala el valor del registro *height*, se pone en alta su línea *max* y la unidad central de control reinicia su cuenta.

Las señales de sincronización de los bloques *Ordena_filas* y *Filtrado_unidim* son descritas en la tabla 2.1:

Señal de control	Indicación	Condición de habilitación
hab_col	Avance de columna en el bloque de Filtrado unidimensional	hab_opj='1'
inicio_fila	Posicionamiento del SE sobre borde Izquierdo de la imagen	opj=00h
fin_fila	Posicionamiento del SE sobre borde Derecho de la imagen	opj_max='1'
hab_fila	Cambio de fila en el bloque Ordena Filas	hab_opi='1'
last_row	Posicionamiento del SE sobre el borde inferior de la imagen	opi_max='1'

TABLA 2.1.- DESCRIPCIÓN DE LAS SEÑALES DE SINCRONIZACIÓN

2.4.2.- BANDERAS DE CONTROL DE TRANSFERENCIA DE DATOS

Establecen un protocolo de comunicación para la transferencia de datos entre la memoria y la interfaz E/S, además forman parte de la palabra de control de cambios de estado. Estas banderas se agrupan en un registro de entrada (ver tabla 2.2) y uno de salida (ver tabla 2.3).

Bit	Nombre	Función
0	*Envio_dato*	Indica que hay nuevo dato en el registro de entrada
1	*R_read*	Indica que el resultado ha sido leído
2	*RST*	Reinicio del sistema

TABLA 2.2.- CONTENIDO DEL REGISTRO DE BANDERAS DE ENTRADA

Bit	Nombre	Función
0	*dato_recibido*	Indica que el dato ha sido escrito en memoria
1	*prep_para_recib*	Indica que el sistema está apto para recibir información
2	*R_listo*	Indica que el resultado está listo para ser leído

TABLA 2.3.- CONTENIDO DEL REGISTRO DE BANDERAS DE SALIDA

Cuando la bandera *prep_para_recib* se encuentra en alta, el sistema está preparado para almacenar un nuevo dato en la memoria, entonces la interfaz E/S coloca un nuevo dato en el bus de datos de entrada, lo cual se lo indica al bloque de control con la bandera *envio_dato* en alta; cuando esto ocurre se concreta la entrada del dato a la memoria. Luego del almacenamiento del dato, el bloque de control pone en alta la bandera *dato_recibido*, esta bandera le ordena a la interfaz E/S que ponga en baja la bandera *envio_dato*.

En el momento que el sistema termina de procesar un nuevo dato de la imagen resultado la bandera *R_listo* se pone en alta, la interfaz E/S almacena el dato

en un registro y señala la lectura del resultado mediante la bandera *R_read*, está bandera se activa por un instante y después se desactiva.

2.4.3.- MAQUINA DE ESTADOS

Todo el proceso de una imagen puede ser dividido en cuatro pasos, los cuales fueron ideados con el fin de facilitar la asignación de las señales de control y ayudar a la verificación de la secuencia del proceso. Estos periodos son controlados por una máquina de estados, cuyos cambios dependen de las banderas de control y los valores de los contadores de filas y columnas. Mediante las banderas de control, el sistema puede realizar la transmisión de datos con dispositivos que trabajan a diferentes velocidades, y sincronizar el flujo de datos.

Los pasos seguidos por el sistema (ver figura 2.21) son:

1) Almacenamiento del SE
2) Almacenamiento de las 2 primeras filas de la imagen
3) Almacenamiento de los píxeles de la imagen y (en simultaneo) obtención de los resultados de la imagen
4) Obtención de las últimas filas del resultado

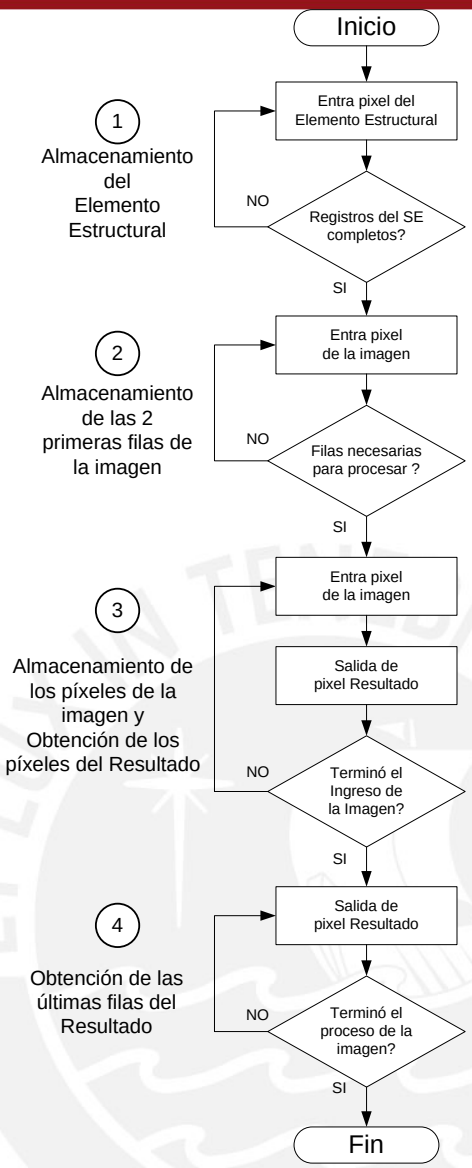
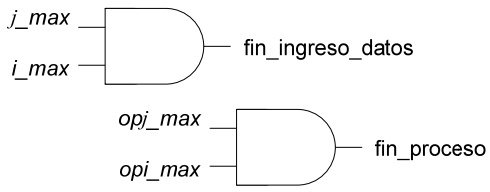


FIGURA 2.21.- DIAGRAMA DE FLUJO DEL PROCESO DE UNA IMAGEN

En la arquitectura de la máquina de estados cada paso se subdivide en dos estados, un estado de espera (par) y un estado de transferencia (impar). Durante la entrega de datos desde la interfaz E/S hacia la memoria, la unidad de control alterna entre un estado de espera y un estado de transferencia, de manera cíclica.

En la tabla 2.4 se describen los estados del bloque de control y se indican los valores que toman los bits que conforman la palabra de control de cambios de

estado (ver figura 2.22), con su respectiva etiqueta. Las habilitaciones de cuenta se dan en el instante que hay un cambio de un estado de espera a un estado de transferencia.



RST	envio_dato	R_read	R_listo	SE_max	im_max	fin_ingreso_datos	fin_proceso

FIGURA 2.22.- PALABRA DE CONTROL DE CAMBIOS DE ESTADO

Paso en el Proceso	Estado	Descripción	Palabra de Control de cambios de estado		
			Etiqueta	Valor	Habilita cuenta
			rst	1XXXXXXX	
1	S0	Espera datos del SE	*c_0*	00XXXXXX	
			c_0a1	01XXXXXX	*hab_SE*
	S1	Habilitación de escritura en el SE	*c_1*	01XXXXXX	
			c_1a0	00XX0XXX	
			c_1a2	00XX1XXX	
2	S2	Espera datos de la imagen	*c_2*	00XXXXXX	
			c_2a3	01XXXXXX	*hab_j*
	S3	Habilitación de escritura en la RAM	*c_3*	01XXXXXX	
			c_3a2	00XXX0XX	
			c_3a4	00XXX1XX	
3	S4	Espera datos de la imagen y lectura de resultado	*c_4*	00XXXXXX	
			c_4a5	011XXXXX	*hab_j ,hab_opj*
	S5	Habilitación de escritura en la RAM y Obtención de un píxel del resultado	*c_5*	0X1XXXXX	
			c_5a4	0001XX0X	
			c_5a6	0001XX1X	
4	S6	Espera lectura de resultado	*c_6*	0X0XXXXX	
			c_6a7	0X1XXXXX	*hab_opj*
	S7	Obtención de un píxel resultado	*c_7*	0X1XXXXX	
			c_7a6	0X01XXX0	
			c_7aZ	0X01XXX1	

TABLA 2.4.- DESCRIPCIÓN DE SEÑALES DE CAMBIOS DE ESTADO

Se tiene un estado de inicio (SA), al cual se cambia cuando la bandera RST se pone en alta (ver figura 2.23), luego que RST se pone en baja se cambia del estado SA al estado S0. En el estado SZ solo se espera que el sistema se reinicie mediante el la bandera de RST.

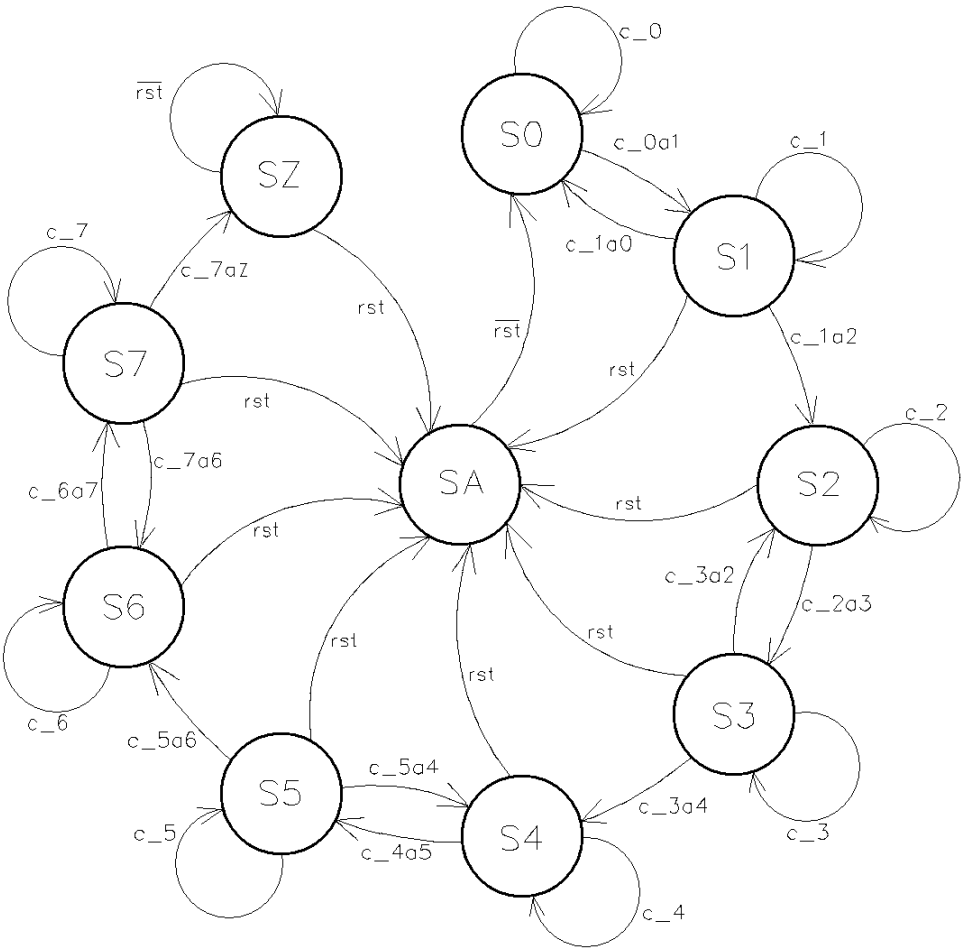


FIGURA 2.23.- DIAGRAMA DE ESTADOS DE LA UNIDAD DE CONTROL

2.5.- DISEÑO DE LOS BLOQUES ARITMÉTICOS

Se tienen cuatro bloques aritméticos que son independientes de las señales del bloque de control, cada uno de ellos realiza una operación específica basándose en la segmentación de sus procesos (*pipeline*).

Las operaciones de suma y resta entre los píxeles de la imagen (*A*) y los píxeles del SE (*B*) indicadas en las fórmulas de dilación y erosión (sección 1.2, fórmulas 3 y 6) se realizan en el bloque Sumador Morfológico (*DILA_SUM.VHD*, ver anexo A.4.1) y en el bloque Restador morfológico (*ERO_SUM*.VHD, ver anexo A.4.2); cada uno de estos bloques está compuesto por sumadores con acarreo, registros, comparadores y multiplexores. En el diseño toman en cuenta las siguientes consideraciones (ver figura 2.24):

1. Cuando el valor de un píxel de *B* es cero, el resultado de la suma es el mínimo valor posible 0, mientras que el resultado de la resta es el máximo posible 255.

2. Si un píxel de *B* es diferente de cero y el valor de uno de *A* es cero (parte del fondo de la imagen), entonces el resultado de la operación es cero.

3. Los resultados deben mantenerse en el rango de 0 a 255.

$$\text{Suma} = \begin{cases} 0 & , A = 0 \ \text{v} \ B = 0 \\ A+B & , 0 < A+B < 255 \\ 255 & , A + B \geq 255 \end{cases}$$

$$\text{Resta} = \begin{cases} 0 & , A - B \leq 0 \\ A-B & , 0 < A-B < 255 \\ 255 & , B=0 \end{cases}$$

FIGURA 2.24.- CONDICIONES DEL RESULTADO DEL SUMADOR Y RESTADOR MORFOLÓGICO

En el diseño del bloque Máxima Suma y el bloque Mínima Resta se plantean dos alternativas de descripción en VHDL, comportamental y estructural. Se analiza el rendimiento de ambas alternativas en función al número de celdas lógicas (LC) que utiliza su implementación sobre un FPGA y la máxima frecuencia de reloj alcanzable.

2.5.1.- SUMADOR MORFOLÓGICO

La primera etapa del circuito lo conforman un sumador con acarreo y dos comparadores que determinan si un píxel de *A* o de *B* es nulo, y en la segunda etapa el resultado de la suma entra a un multiplexor, cuyas líneas de control son el bit de acarreo (*c*) y el bit resultado de la comparación de *A* con cero (*A0*); si el bit *c* se pone en alta entonces la salida del primer multiplexor se limita a 255, pero si *A0* está en alta entonces la salida obtenida es cero, caso contrario se obtiene como resultado la suma de A y B. Finalmente la salida del primer multiplexor entra a un segundo que es controlado por el bit de comparación de *B* con cero (*B0*); solo si este bit se pone en alta entonces el resultado final es

nulo, de lo contrario el resultado final es el obtenido por el primer multiplexor (ver figura 2.25).

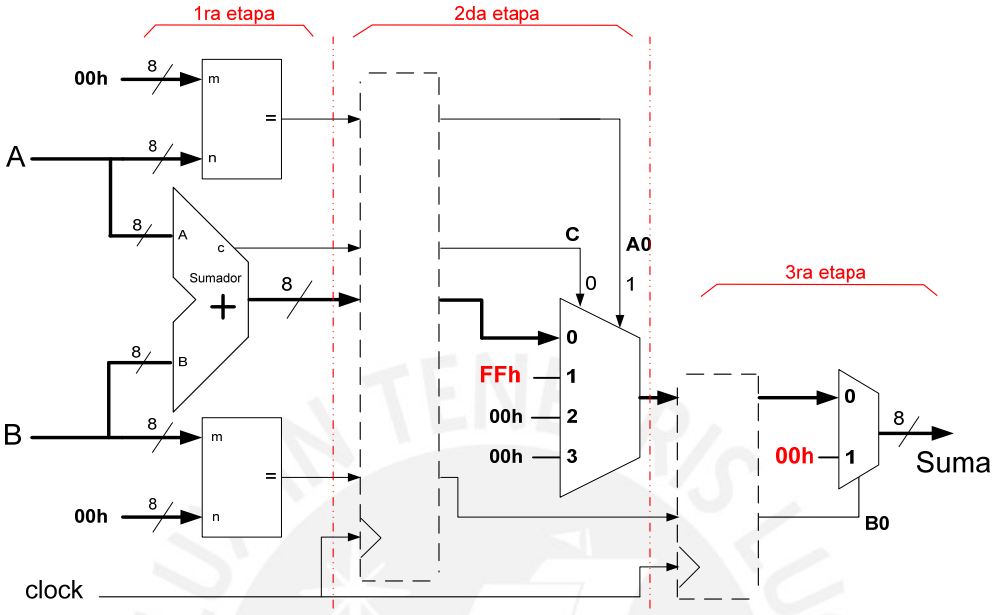


FIGURA 2.25.- DIAGRAMA LÓGICO DE UN SUMADOR MORFOLÓGICO

Cabe indicar que en cada etapa se registran las salidas con el fin de mantener la sincronía del sistema, así el circuito funciona con un *pipeline* de latencia 2 (ver figura 2.26).

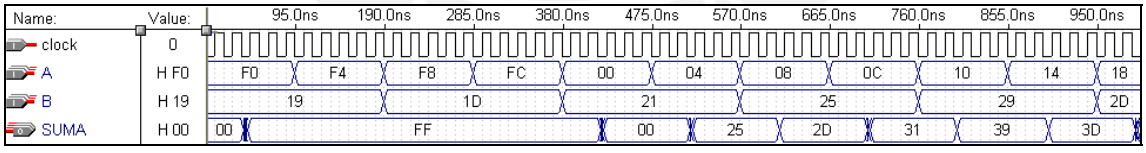


FIGURA 2.26.- SIMULACIÓN DEL BLOQUE *DILA_SUM*

2.5.2.- RESTADOR MORFOLÓGICO

Su diseño es muy similar al del sumador morfológico (ver figura 2.27), en la primera etapa del circuito se tiene un restador con un bit de desborde (*c*). En caso que el bit *c* o la señal que indica que es cero (*A0*) estén en alta, la salida del primer multiplexor será cero. Cuando el bit *B0* se encuentra en alta, no se toma en cuenta el resultado del primer multiplexor y el resultado final es 255. En el momento que el valor de resta es 255 el bloque Mínima Resta descarta su valor porque siempre habrá un número menor que 255; el único caso en el que el resultado de un bloque Mínima Resta sea 255, se da cuando todos los coeficientes del elemento estructural son iguales a 0 (ver figura 2.28).

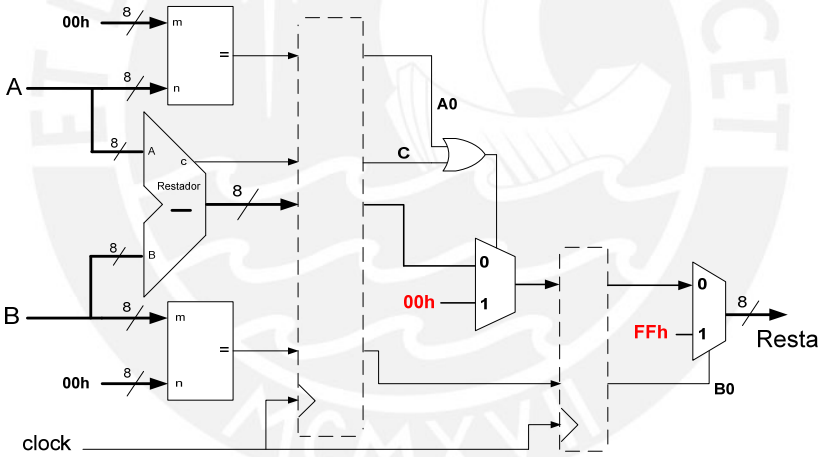


FIGURA 2.27.- DIAGRAMA LÓGICO DE UN RESTADOR MORFOLÓGICO

Cabe indicar que en cada etapa se registran las salidas con el fin de mantener la sincronía del sistema, así el circuito funciona con un *pipeline* de latencia 2 (ver figura 2.28).

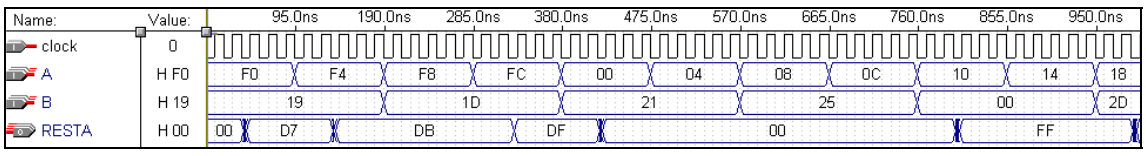


FIGURA 2.28.- SIMULACIÓN DEL BLOQUE *ERO_SUM*

2.5.3.- BLOQUE MÁXIMA SUMA

2.5.3.1.- DESCRIPCIÓN COMPORTAMENTAL

La descripción comportamental es un punto de partida para diseñar un circuito que obtenga el mayor valor de la comparación de tres números (*a, b* y *c*) (ver figura 2.29), porque resulta más rápido y sencillo desde el punto de vista de un programador.

```
process(clock)
  begin
    if clock'event and clock='1' then
      c_t <= c;

      if a > b then mayor_t <= a;
      else mayor_t <= b; end if;

      if mayor_t > c_t then mayor <= mayor_t;
      else mayor <= c_t; end if;

    end if;
end process;
```

Donde:
a, b, c : Son los valores a comparar
mayor : Resultado final, que indica el valor máximo
mayor_t : Resultado parcial
c_t : Entrada "c" registrada

FIGURA 2.29.- DESCRIPCIÓN COMPORTAMENTAL EN VHDL DEL BLOQUE MÁXIMA SUMA

El algoritmo consiste en comparar primero dos valores, de los cuales el mayor valor obtenido se registra en una variable; esta variable registrada (*mayor_t*) se compara con el tercer valor registrado (*c_t*) obteniéndose el mayor de ellos como resultado final. El circuito funciona con un *pipeline* de latencia 2.

Compilando este bloque en modo de síntesis *NORMAL* se obtuvo un consumo de 70 LCs y una frecuencia de reloj máxima de 34.84Mhz. Mientras que en el modo de síntesis *FAST* el consumo de LCs fue de 68, y la máxima de frecuencia de reloj fue 38.02Mhz.

2.5.3.2.- DESCRIPCIÓN ESTRUCTURAL

Debido a que un circuito comparador de palabras combinacional genera un retardo considerable, se sugiere como alternativa la descripción estructural. La comparación de los valores de entrada consiste en módulos que segmentan las palabras en sus bits mas significativos y menos significativos, los cuales son comparados por separado y luego componen sus resultados. Se parte desde la descripción de módulos comparadores de bits (ver figura 2.30).

M_in	A	B	F (A>B)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

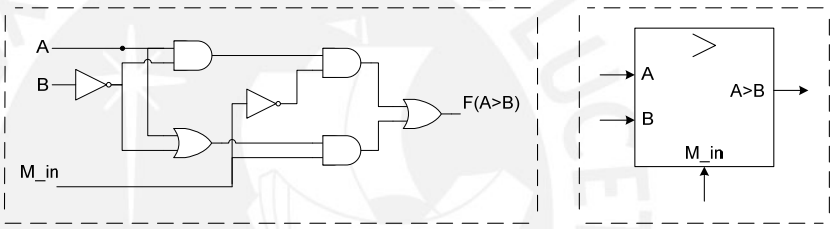


FIGURA 2.30.- DIAGRAMA LÓGICO DE UN BLOQUE COMPARADOR DE UN BIT, CON BIT DE COMPARACIÓN ANTERIOR (MAYOR)

El bloque comparador de 1 bit constituye una unidad básica dentro de los comparadores de palabras (ver figura 2.31). Se tienen 2 números *A* y *B* de 'm' bits, por ello se emplean 'm' comparadores en los cuales sus entradas son un bit de cada número y la entrada *M_in* es el resultado de la comparación de los bits menos significativos.

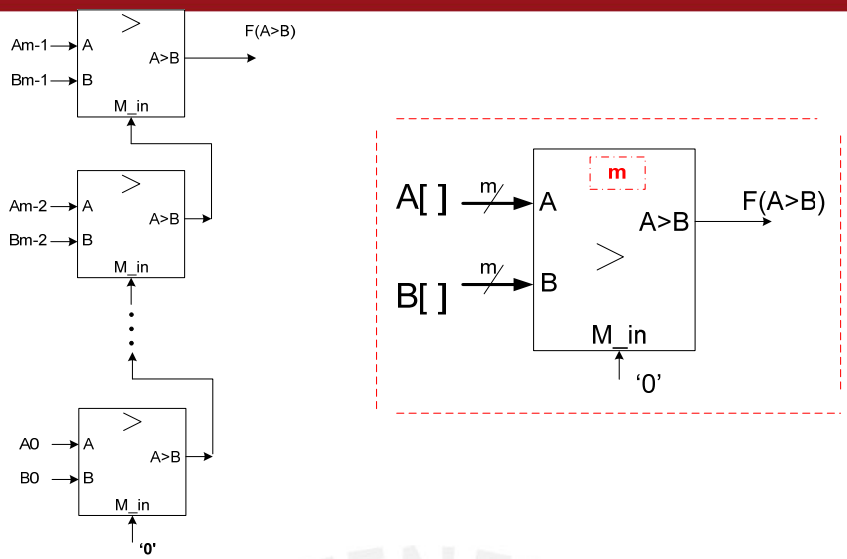


FIGURA 2.31.- DIAGRAMA ESTRUCTURAL DE UN COMPARADOR DE PALABRAS, CON BIT DE COMPARACIÓN ANTERIOR

Un bloque que obtiene el mayor valor entre dos números está constituido por un comparador y un multiplexor, en el cual el bit resultado de la comparación se encarga de controlar la selección del multiplexor, cuyas entradas son los números en cuestión. Cuando se incrementa el número de bits de la palabra, el resultado de la comparación demora mayor tiempo, por ello es preciso segmentar los números en sus bits más significativos y menos significativos, los cuales son comparados por separado y sus resultados se componen con el fin de obtener el resultado final de la comparación (ver figura 2.32). Además se agregan registros a la salida de los comparadores para sincronizar las señales con el sistema, esto le agrega latencia al proceso pero le permite alcanzar mayores frecuencias de reloj [18].

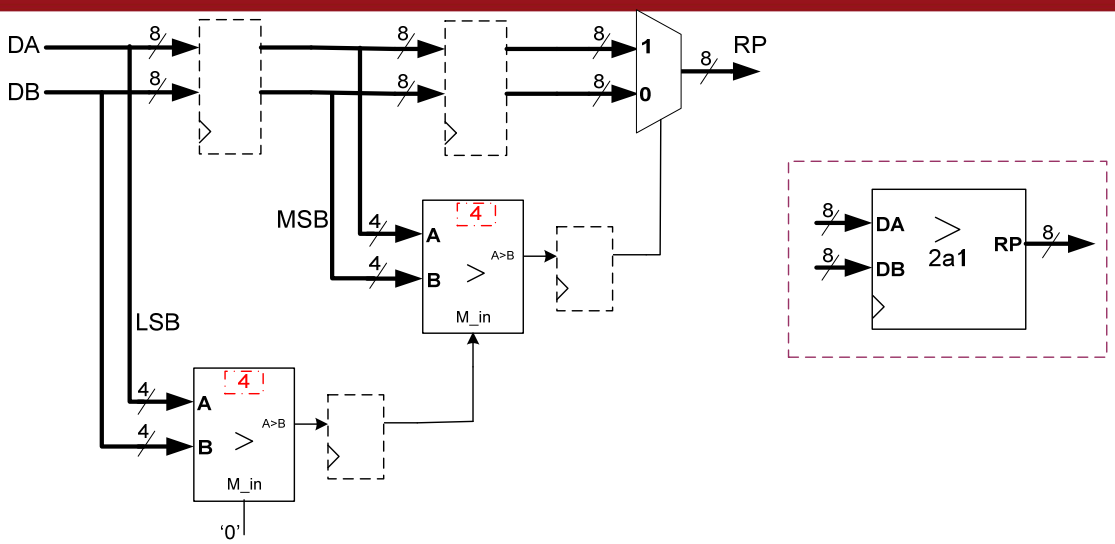


FIGURA 2.32.- DIAGRAMA LÓGICO DE UN CIRCUITO QUE OBTIENE EL MAYOR VALOR ENTRE DOS NÚMEROS

Utilizando dos de estos módulos comparadores se extiende el circuito a la comparación de tres números diferentes (ver figura 2.33). Se compara el resultado (*RP*) de la comparación de los dos primeros (A y B) con el tercer número (*C*) y se obtiene como resultado final el mayor valor entre los tres (ver anexo A.4.3).

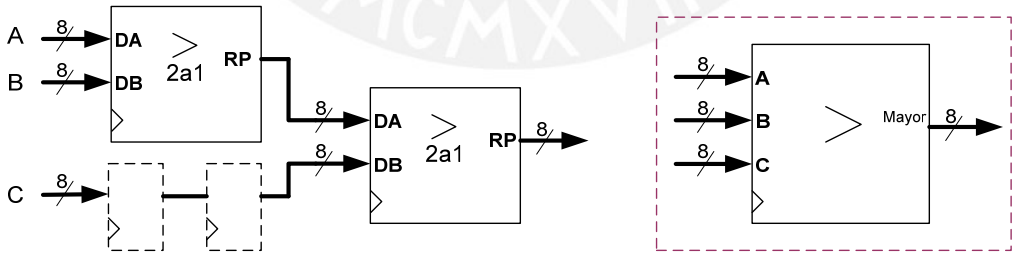


FIGURA 2.33.- DISEÑO DEL BLOQUE MÁXIMA SUMA A PARTIR DE UN BLOQUE QUE OBTIENE EL MAYOR VALOR DE TRES NÚMEROS

El resultado de la compilación indica un consumo de 106 celdas lógicas en modo de síntesis *NORMAL* con una frecuencia de reloj máxima de 59.88Mhz.

En modo de síntesis *FAST* se obtuvo el mismo consumo de celdas lógicas pero con una frecuencia de reloj máxima de 84.03Mhz (ver figura 2.34).

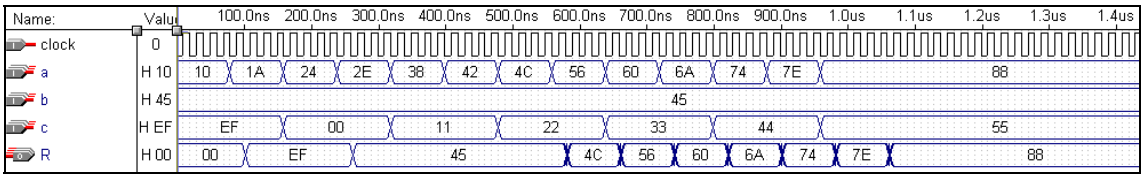


FIGURA 2.34.- SIMULACIÓN DEL MÓDULO *MAYOR_PIPE*

2.5.4.- BLOQUE MÍNIMA RESTA

2.5.4.1.- DESCRIPCIÓN COMPORTAMENTAL

Su descripción sigue el mismo esquema algorítmico que el bloque máxima suma, en este caso se obtiene el menor número entre tres números distintos (a, b, c) (ver figura 2.35).

```
process(clock)
  begin
    if clock'event and clock='1' then
       c_t <= c;

       if a > b then menor_t <= a;
       else menor_t <= b; end if;

       if menor_t > c_t then menor <= menor_t;
       else menor <= c_t; end if;

    end if;
end process;
```

Donde:
a, b, c : Son los valores a comparar
menor : Resultado final, que indica el valor mínimo
menor_t : Resultado parcial
c_t : Entrada "c" registrada

FIGURA 2.35.- DESCRIPCIÓN COMPORTAMENTAL EN VHDL DEL BLOQUE MÍNIMA RESTA

El resultado de la compilación indica que este bloque utiliza el mismo número de celdas lógicas que el bloque máxima suma con descripción comportamental, tanto en modo de síntesis *NORMAL* como *FAST*. Sin embargo indica valores mayores de frecuencia de reloj, 40.48MHZ en modo *NORMAL* y 47.39Mhz en modo *FAST*.

2.5.3.2.- DESCRIPCIÓN ESTRUCTURAL

Se mantiene la misma estructura de comparación segmentada de números que presenta el bloque máxima suma (ver figuras 2.31 y 2.32), pero se cambia la lógica de comparación de bits (ver figura 2.36) por uno que indique que A es menor que B y que tenga un bit de comparación anterior; con este cambio se compone un bloque que obtiene el menor valor de tres números (ver figura 2.37).

M_in	A	B	F (A<B)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

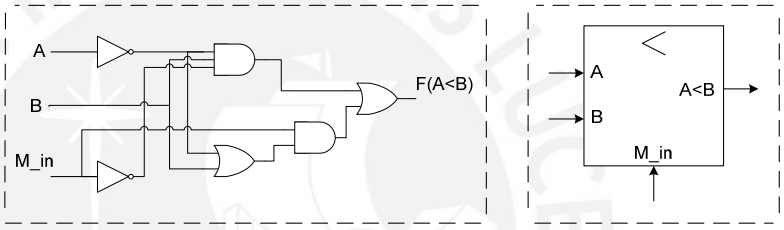


FIGURA 2.36.- DIAGRAMA LÓGICO DE UN COMPARADOR UN DE BIT, CON BIT DE COMPARACIÓN ANTERIOR (MENOR)

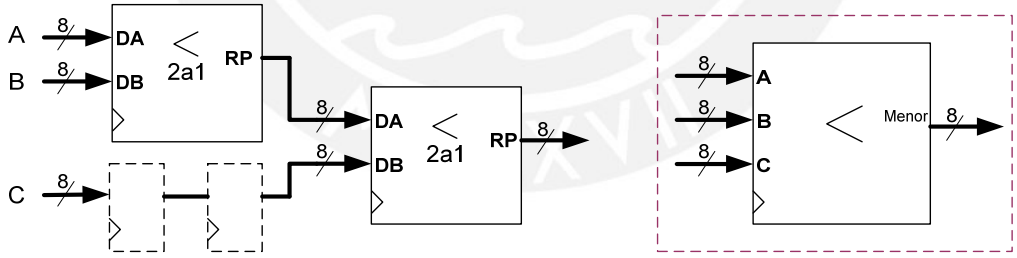


FIGURA 2.37.- DISEÑO DEL BLOQUE MÍNIMA RESTA A PARTIR DE UN BLOQUE QUE OBTIENE EL MENOR VALOR DE TRES NÚMEROS

La compilación del módulo descrito (*MENOR_PIPE.VHD*) (ver anexo A.4.4), indicó un consumo de 106 celdas lógicas en modo de síntesis *NORMAL* y *FAST*, con una frecuencia de reloj máxima de 55.24Mhz en modo *NORMAL* y 90.90 Mhz en modo *FAST* (ver figura 2.38).

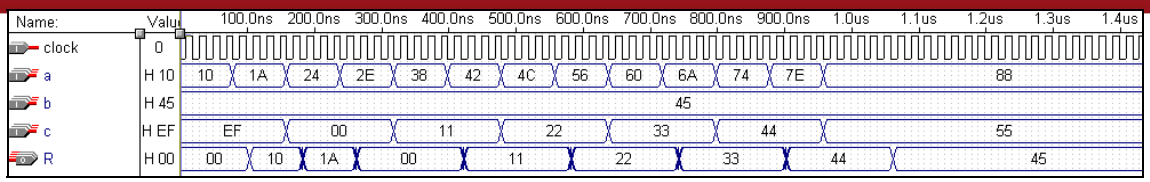


FIGURA 2.38.- SIMULACIÓN DEL MÓDULO *MENOR_PIPE*

Con la descripción estructural se obtienen frecuencias de reloj muy superiores a las obtenidas en descripción comportamental y prácticamente se duplican. Sin embargo, los módulos descritos estructuralmente consumen mayor cantidad de celdas lógicas (ver tabla 2.5).

Bloque	Tipo de Descripción	Tipo de síntesis	Celdas Lógicas (LC)	Frecuencia Máxima (Mhz)	Latencia de clocks
Máxima Suma	Comportamental	NORMAL	70	34.84	2
		FAST	68	38.02	2
	Estructural	NORMAL	106	59.88	4
		FAST	106	84.03	4
Mínima Resta	Comportamental	NORMAL	70	40.48	2
		FAST	68	47.39	2
	Estructural	NORMAL	106	55.24	4
		FAST	106	90.90	4

TABLA 2.5.- COMPARACIÓN DEL RENDIMIENTO DE LOS BLOQUES MÁXIMA SUMA Y MÍNIMA RESTA SEGÚN EL TIPO DE DESCRIPCIÓN

Es determinante alcanzar frecuencias de reloj mayores a 40 Mhz para la implementación de la arquitectura en la tarjeta Constellation [11], y las arquitecturas Máxima suma y Mínima Resta que son descritos estructuralmente sobrepasan esta condición.

2.6.- ESPECIFICACIONES DE LA INTERFAZ E/S

La interfaz E/S está diseñada para comunicar la arquitectura implementada con la memoria de la PC a través del Bus ISA [19]. La interfaz posee un bloque de control que sincroniza la transferencia de datos en base a las banderas de control, con lo cual no es necesaria la sincronización del reloj del sistema con las señales del Bus ISA. Durante la transferencia de datos, la Interfaz E/S cumple con los procesos de acceso a la memoria interna, además de la lectura y la escritura de los registros de banderas (ver figura 2.39).

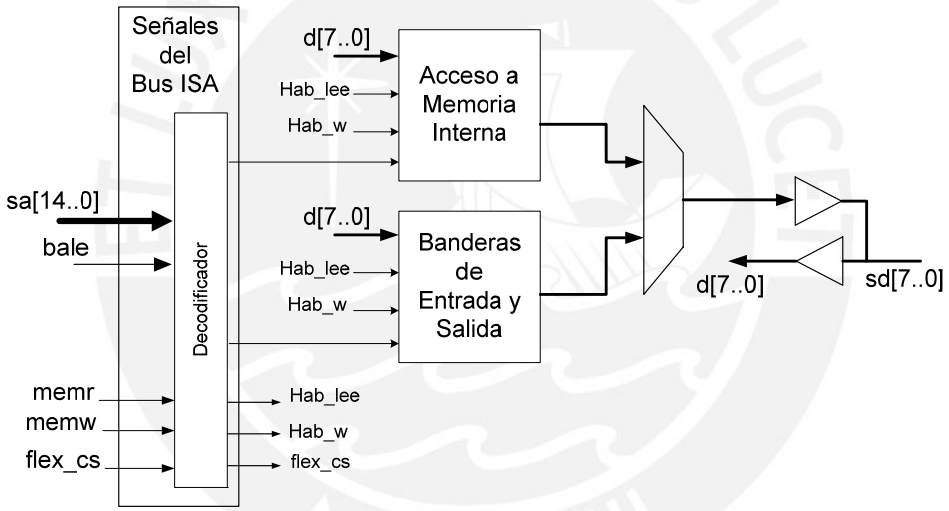


FIGURA 2.39.- INTERFAZ E/S AL BUS ISA

El sistema solo requiere la administración de cuatro registros (ver tabla 2.6) para el procesamiento de una imagen, sin embargo permite la visualización de otros registros que pueden ser usados en la verificación del proceso.

Registro	Longitud de Bits	Tipo de Registro	Descripción
Dato_in	8	Escritura	Registro de entrada de Datos
Resultado	16	Lectura	Salida de Resultados
Controla	4	Escritura	Registro de Banderas de Entrada
Senales	6	Lectura	Registro de Banderas de Salida

TABLA 2.6.- REGISTROS DEL SISTEMA PARA EL PROCESO DE UNA IMAGEN

Las señales de control ISA (ver tabla 2.7) de lectura (*memr*), escritura (*memw*), selector de chip (*flex_cs*) y las líneas de dirección son almacenados en registros con el fin de ser sincronizados con el reloj de la tarjeta Constellation [11] y puedan ser utilizados por los módulos de direccionamiento. Las 4 líneas menos significativas del bus de direcciones son decodificadas para establecer la dirección de memoria en la cual se escriben o se leen los registros de la interfaz E/S.

Señal	Tipo	Descripción
NOWS	Salida	Indica al CPU que el dispositivo no necesita tiempos de espera para la transferencia de datos
MEMCS16	Salida	Indica que el dispositivo soporta transferencias de 16 bits. Se activa en baja.
SD[15..0]	Entrada / Salida	Bus bidireccional de datos del bus ISA
SA[14..0]	Entrada	Bus de direcciones del bus ISA. Solo es necesaria la decodificación de las 4 líneas menos significativas
BALE	Entrada	Se utiliza para indicar al sistema que hay una dirección válida en el bus
MEMR	Entrada	Señal de lectura. Habilita lectura de registros. Activa en baja
MEMW	Entrada	Señal de escritura. Habilita escritura en registros. Activa en baja
Flex_cs	Entrada	Selector de chip. Indica al sistema que está siendo direccionado por el bus ISA.

TABLA 2.7.- SEÑALES ISA GENERADAS POR EL SISTEMA

El módulo denominado *control_banderas.vhd* (ver anexo A.6.1) se encarga de traducir las señales de control del Bus ISA y puedan ser entendidas por los diferentes procesos. Con ello, es posible controlar las banderas de entrada (ver figura 2.40) del sistema utilizando el bus de datos, sin embargo esto implica realizar accesos extra al bus ISA, lo cual solo es útil cuando se realizan pruebas funcionales del sistema con verificación de registros paso a paso, esto se da cuando la señal de control *MODO* se pone en baja al inicio de un

proceso, con lo cual la escritura en el registro de banderas de entrada del sistema se realiza desde la PC a través del bus de datos ISA.

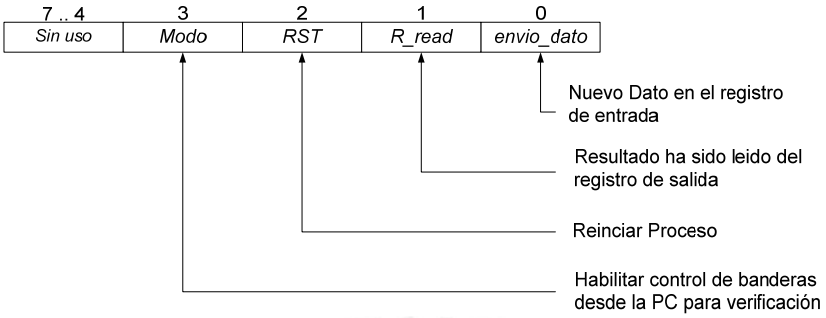


FIGURA 2.40.- REGISTRO DE BANDERAS DE ENTRADA

Para verificar el estado del sistema y determinar si está apto para recibir o enviar datos, se lee el registro de banderas de salida (ver figura 2.41).

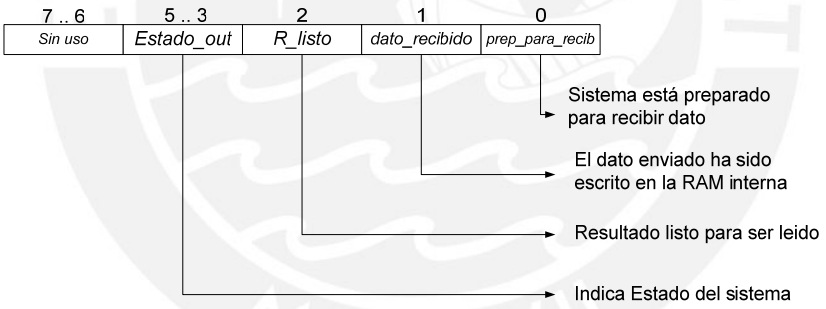


FIGURA 2.41.- REGISTRO DE BANDERAS DE SALIDA

2.7.- RESULTADOS DE LAS COMPILACIONES

Se realizan las compilaciones de la arquitectura diseñada y sus diferentes partes utilizando las herramientas de Altera MAX+plus II versión 10.1 en la cual se emplean los tipos de síntesis *NORMAL* y *FAST* [17] y la versión 3.0 de Quartus II con la síntesis *NORMAL FIT* [17]. A partir de los resultados obtenidos se analizan los parámetros de: número de celdas lógicas (LC), bits de memoria y máxima frecuencia de reloj alcanzada. Este análisis se da en diferentes familias de FPGAs para imágenes de 256x256 píxeles, luego se observa la variación de los parámetros para distintos tamaños de imágenes sobre el FPGA EPF10K100EQC240-3 (FLEX10KE) [8] y finalmente se mide el índice de ocupación en la implementación realizada por cada bloque que conforma la arquitectura para imágenes de 256x256 píxeles.

2.7.1.- COMPILACIÓN EN DISTINTOS FPGA PARA IMÁGENES DE TAMAÑO 256X256 PÍXELES

Las compilaciones se hicieron para los FPGA EP1K50TC144 [9], EPF10K100EQC240 [8] y EP1S10F780C [10] de las familias ACEX1K, FLEX10KE y Stratix respectivamente; esta última solo se encontró disponible en Quartus II con los grados de velocidad -5, -6 y -7. La tabla 2.8 resume los resultados obtenidos para los diferentes grados de velocidad con los que se disponen, y en las comparaciones se toman en cuenta los diferentes tipos de síntesis (ver figuras 2.42, 2.44, 2.45).

FPGA (Familia)	Bits de memoria	% Bits de memoria	Tipo de Síntesis	Celdas Lógicas (LC)	% LC	Grado de Velocidad	Máxima Frecuencia (MHz)
EPF10K100EQC240 (FLEX10KE)	8192	16.66	*Normal*	2157	43.21	-1	81.96
						-2	56.49
						-3	40
			Fast	1970	39.46	-1	94.33
						-2	72.99
						-3	58.47
			Normal Fit	2117	42.41	-1	135.14
						-2	96.15
						-3	80.65
EP1K50TC144 (ACEX1K)	8192	33.33	*Normal*	2157	74.89	-1	82.63
						-2	66.66
						-3	48.30
			Fast	1970	68.40	-1	111.11
						-2	88.49
						-3	64.51
			Normal Fit	2117	73.50	-1	153.85
						-2	133.33
						-3	97.74
EP1S10F780C (Stratix)	8507	0.92	*Normal Fit*	1562	14.77	-5	258.60
						-6	245.58
						-7	177.78

TABLA 2.8.- RESULTADOS DE COMPILACIÓN DE LA ARQUITECTURA PARA DISTINTOS FPGA

Se aprecia una menor ocupación de LCs en síntesis *FAST*, mientras que en modo *NORMAL* y *NORMAL FIT* los valores se aproximan; el consumo es el mismo en las familias FLEX10KE y ACEX1K (ver figura 2.42) y es mínimo en la familia Stratix, en modo de síntesis *NORMAL FIT*. Los bits de memoria utilizados son independientes al tipo de síntesis, siendo mayor el consumo para la familia Stratix (ver figura 2.43).

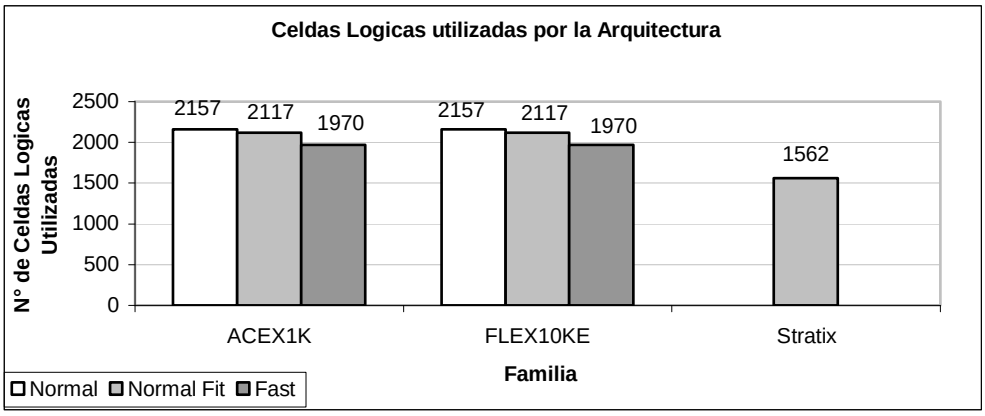


FIGURA 2.42.- COMPARACIÓN DE CELDAS LÓGICAS OCUPADAS EN DISTINTAS FAMILIAS DE FPGA

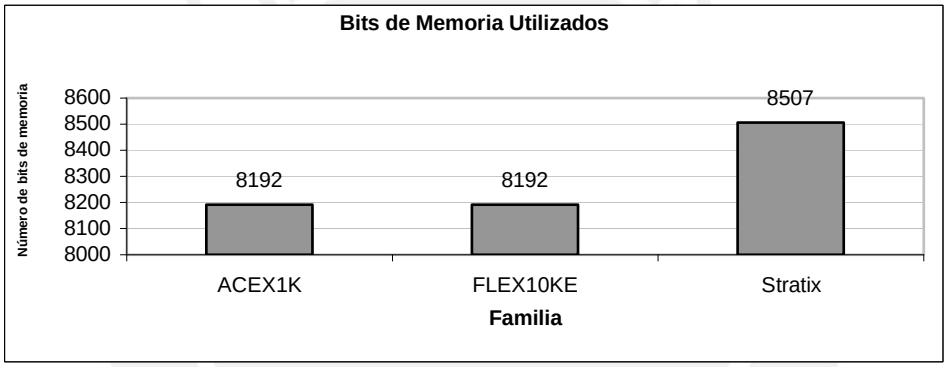


FIGURA 2.43.- COMPARACIÓN DE BITS DE MEMORIA UTILIZADOS EN DISTINTAS FAMILIAS DE FPGA

Los FPGA de la familia Stratix con grado de velocidad -5 obtienen frecuencias de reloj superiores a las obtenidas con otros de grado -6 y -7, además de sobrepasar a las frecuencias obtenidas por los FPGA de las familias ACEX1K y FLEX10K (ver figura 2.44).

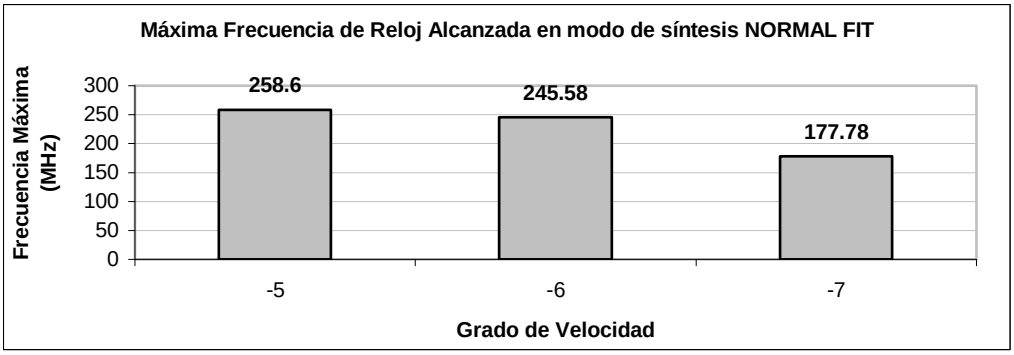
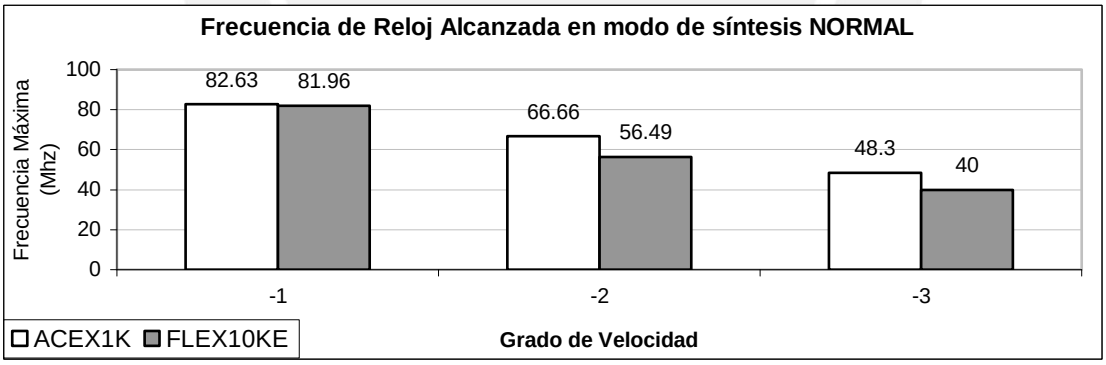
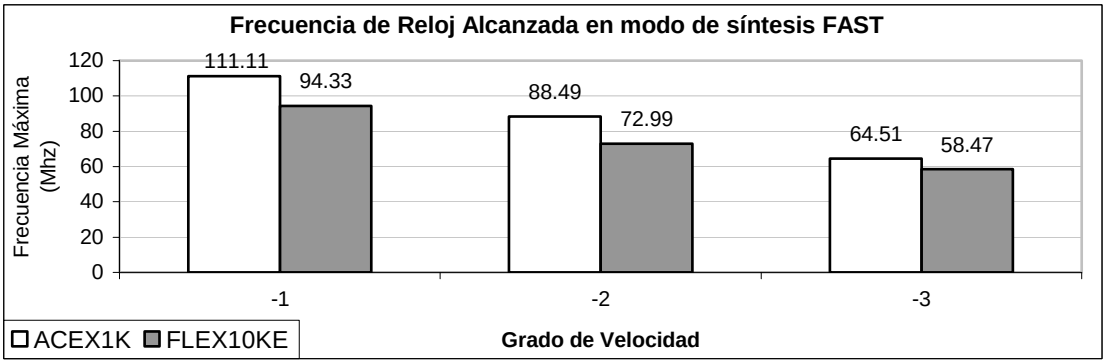


FIGURA 2.44.- COMPARACIÓN DE FRECUENCIA MÁXIMA EN LA FAMILIA STRATIX PARA DISTINTOS GRADOS DE VELOCIDAD

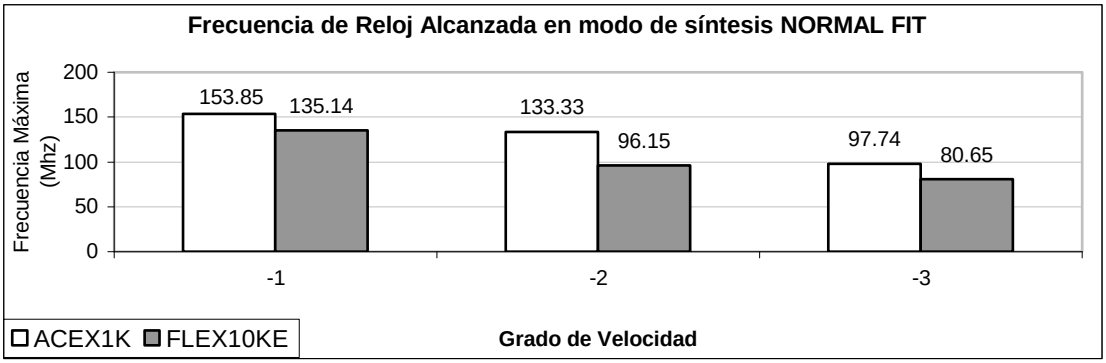
Al comparar el EPF10K100EQC240 (FLEX10KE) y el EP1K30TC144 (ACEX1K), se obtuvieron mayores frecuencias para los grados de velocidad -1, siendo siempre la familia ACEX1K la que presenta los más veloces (ver figura 2.45). En síntesis *NORMAL FIT* se alcanzaron mayores frecuencias de reloj y en *NORMAL* se obtuvieron las menores. En modo *NORMAL*, las frecuencias obtenidas son similares en ambos tipos de familias.



(A)



(B)



(C)

FIGURA 2.45.- COMPARACIÓN DE FRECUENCIA MÁXIMA ENTRE LAS FAMILIAS FLEX10KE Y ACEX1K PARA DISTINTOS GRADOS DE VELOCIDAD

2.7.2.- COMPILACIÓN PARA DISTINTOS TAMAÑOS DE IMAGEN

Se hacen variar los parámetros genéricos que determinan las dimensiones máximas de la imagen a procesar para un FLEX10KE, de esta manera se obtienen los resultados para imágenes de tamaños 32x32, 64x64, 128x128, 256x256, 512x512 y 1024x1024 píxeles (ver tabla 2.9).

La cantidad de bits de memoria utilizados es proporcional al ancho de la imagen (ver figura 2.46).

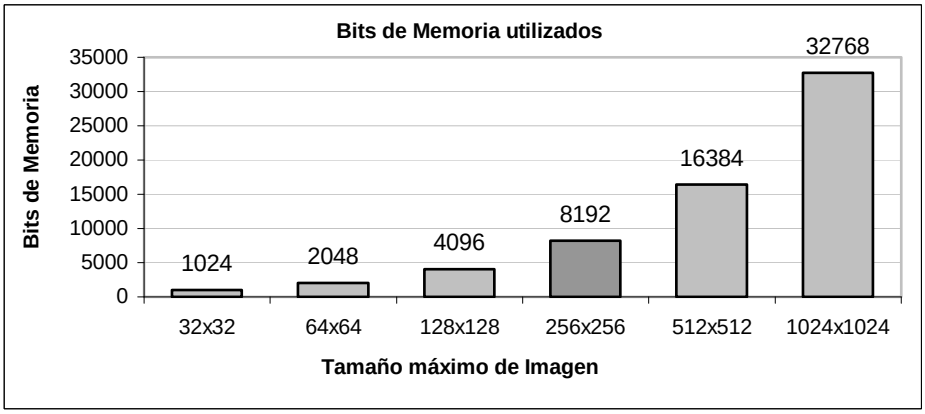


FIGURA 2.46.- COMPARACIÓN DEL NÚMERO DE BITS DE MEMORIA UTILIZADOS PARA DIFERENTES TAMAÑOS MÁXIMOS DE IMAGEN

Tamaño Máximo de Imagen	Bits de Memoria	Tipo de Síntesis	Celdas Lógicas (LC)	% LC	Máxima Frecuencia (MHz)
32x32	1024	*Normal*	2100	42.07	40.00
		Fast	1948	39.02	58.82
		Normal Fit	2095	41.96	81.30
64x64	2048	*Normal*	2116	42.39	36.63
		Fast	1953	39.12	63.69
		Normal Fit	2101	42.09	72.99
128x128	4096	*Normal*	2136	42.79	38.31
		Fast	1964	39.34	52.63
		Normal Fit	2111	42.29	78.74
256x256	8192	*Normal*	2157	43.21	40.00
		Fast	1970	39.46	58.47
		Normal Fit	2117	42.41	80.65
512x512	16384	*Normal*	2173	43.52	36.49
		Fast	1978	39.62	56.81
		Normal Fit	2127	42.60	83.37
1024x1024	32768	*Normal*	2193	43.93	40.00
		Fast	1987	39.80	52.63
		Normal Fit	2134	42.75	76.92

TABLA 2.9.- RECURSOS DEL FPGA UTILIZADOS PARA DISTINTOS TAMAÑOS MÁXIMOS DE IMAGEN

El mayor consumo de LCs se obtuvo con la síntesis *NORMAL*, seguido por la *NORMAL FIT* (ver figura 2.47). La variación del consumo de LCs es mínima cuando se incrementa el tamaño de la imagen, con un promedio de 8 LCs en modos *FAST* y *NORMAL FIT*, y de 19 en *NORMAL*.

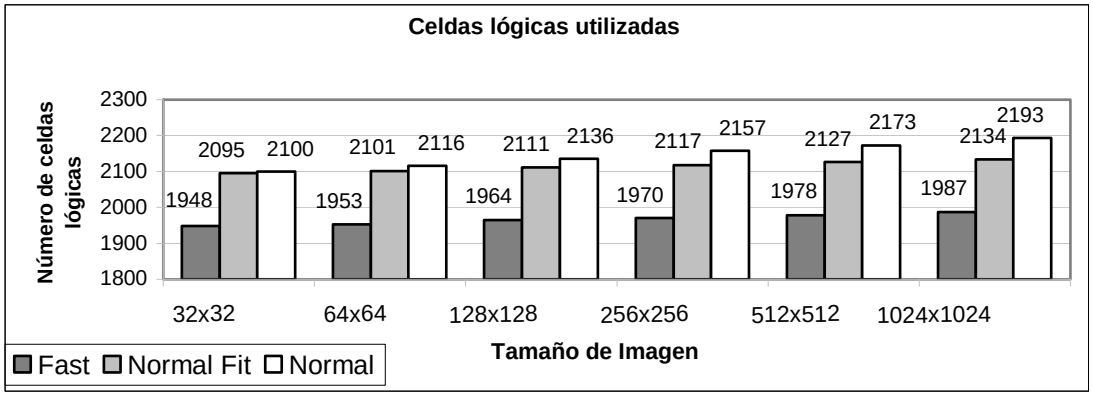


FIGURA 2.47.- COMPARACIÓN DEL PORCENTAJE DE CELDAS LÓGICAS UTILIZADAS PARA DIFERENTES TAMAÑOS MÁXIMOS DE IMAGEN

Para cada tipo de síntesis los resultados indicaron frecuencias de reloj con variación aleatoria en los diferentes tamaños de imagen. Los resultados en síntesis NORMAL FIT indicaron las máximas frecuencias de reloj, con valores que oscilan entre 72.99 y 83.77 MHz (ver figura 2.48).

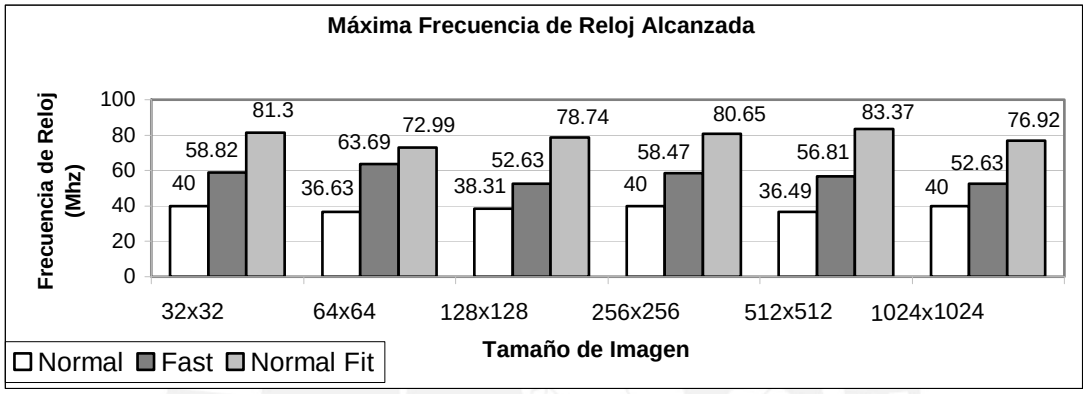


FIGURA 2.48.- COMPARACIÓN DE MÁXIMA FRECUENCIA DE RELOJ ALCANZADA PARA DIFERENTES TAMAÑOS MÁXIMOS DE IMAGEN

2.7.3.- UTILIZACIÓN DE LOS RECURSOS DEL FPGA

Se compilan los diferentes bloques de la arquitectura por separado para un tamaño de imagen 256x256 píxeles para un FLEX10KE con lo cual se verifican los componentes de mayor ocupación y que su máxima frecuencia sobrepase los 40 MHz (ver tabla 2.10).

Bloque	Módulo VHDL	Tipo de Síntesis	Celdas Lógicas (LC)	% (LC)	Máxima Frecuencia (MHz)
Arquitectura con Interfaz E/S	*MM_GRIS.vhd*	*Normal*	2301	46.09	39.68
		Fast	2121	42.49	57.80
		Normal Fit	2225	44.57	77.52
Sincronización de Banderas de control	*control_banderas.vhd*	*Normal*	22	0.44	120.48
		Fast	23	0.46	149.25
		Normal Fit	23	0.46	138.89
Arquitectura sin Interfaz E/S	*MORFOVAR.vhd*	*Normal*	2157	43.21	40
		Fast	1970	39.46	58.47
		Normal Fit	2117	42.41	80.65
Bloque de control, RAM y almacenamiento del SE	*control_datos.vhd*	*Normal*	323	6.47	54.05
		Fast	246	4.93	58.47
		Normal Fit	238	4.77	84.75
Bloque de ordenamiento de filas	*ordena_filas.vhd*	*Normal*	113	2.26	99
		Fast	113	2.26	99
		Normal Fit	114	2.28	106.38
Bloque de filtrado unidimensional	*filtrado_unidim.vhd*	*Normal*	501	10.04	40.65
		Fast	501	10.04	62.5
		Normal Fit	504	10.09	90.09
Bloque Sumador Morfológico	*Dila_sum.vhd*	*Normal*	29	0.58	200
		Fast	23	0.46	200
		Normal Fit	30	0.60	200
Bloque Restador Morfológico	*Ero_sum.vhd*	*Normal*	29	0.58	200
		Fast	23	0.46	200
		Normal Fit	30	0.60	200
Bloque Máxima Suma	*Mayor_pipe.vhd*	*Normal*	106	2.12	59.88
		Fast	106	2.12	84.03
		Normal Fit	109	2.18	109.89
Bloque Mínima Resta	*Menor_pipe.vhd*	*Normal*	106	2.12	55.24
		Fast	106	2.12	90.90
		Normal Fit	109	2.18	107.53

TABLA 2.10.- UTILIZACIÓN DE RECURSOS DE *HARDWARE* Y FRECUENCIA MÁXIMA DE CADA BLOQUE DEL SISTEMA

En modo de síntesis *NORMAL FIT* se obtuvieron mayores frecuencias de reloj y un menor consumo de LCs en *FAST* para la mayoría de los casos.

La implementación del sistema completo (incluyendo la interfaz E/S) en el EPF10K100EQC240-3 se realizo en modo de síntesis FAST, donde el resultado de la compilación indicó las siguientes características:

- Celdas Lógicas usadas: 2121 (42%)
- Unidades de memoria usadas (EAB): 4 de 12
- Bits de Memoria utilizados: 8192 (16.67%)
- Pines Utilizados
  Entrada: 26
  Salida: 3
  Bidireccionales: 16

Dado que el índice de ocupación del FPGA es 42% es posible incluir más módulos de post procesamiento u otro filtro morfológico adicional que permita la obtención de más operaciones morfológicas, sin embargo configurar la arquitectura para imágenes de tamaño mayor limita la capacidad del FPGA para incluir más bloques de procesamiento.

3. ANÁLISIS DE LOS RESULTADOS EXPERIMENTALES

3.1.- INTRODUCCIÓN

La arquitectura diseñada fue implementada y probada en la tarjeta de desarrollo Constellation-E de Nova Engineering [11] que cuenta con un FPGA FLEX10K100EQC240 [8] de grado de velocidad -3, esta tarjeta permite la comunicación con la PC a través del bus ISA. La interacción con el sistema implementado en la tarjeta de desarrollo se realiza mediante un programa de control que cuenta con una interfaz usuario y que fue desarrollado en un lenguaje de programación basado en *C* (ver anexo B) bajo plataforma Windows 98.

La labor principal del programa de control en el proceso de una imagen consiste en sincronizar el envío de datos desde la PC hacia el sistema con la lectura de los resultados obtenidos, a través del bus ISA (ver figura 3.1). Esta tarea es verificada mediante la lectura de registros de estado que indician la secuencia del proceso llevado a cabo dentro del sistema.

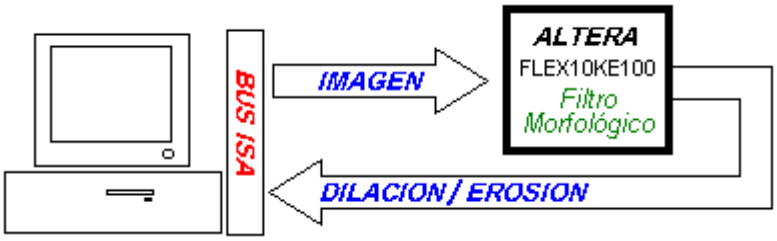


FIGURA 3.1.- PROCESO DE FILTRADO A TRAVÉS DEL BUS ISA

En la comprobación de los resultados obtenidos por el sistema, se desarrollaron algoritmos en MATLAB [12] [20] que hacen las operaciones de dilación y erosión en escala de grises (anexos C.1 y C.2); los resultados

obtenidos en Matlab ratifican que el sistema implementado arroja resultados correctos. Adicionalmente se diseñó un programa en Matlab para medir los tiempos de procesamiento que emplean sus algoritmos de dilación y erosión en escala de grises (ver anexo C.3).

3.2.- DISEÑO DEL PROGRAMA DE CONTROL

El programa de control (CP) gobierna la transferencia de datos entre la PC y el sistema, para ello accede a los registros del sistema que son definidos por la interfaz E/S (ver sección 2.6). De esta manera cumple con la tarea de procesar una imagen, que involucra el envío de datos del SE y de la imagen, hacia la memoria del sistema; en simultaneo los resultados obtenidos son almacenados en la memoria de la PC. Adicionalmente a la tarea de procesar imágenes, el CP permite realizar pruebas de funcionamiento del sistema que verifican paso a paso el proceso de una imagen, además de pruebas de rendimiento que consisten en la medición de tiempos de procesamiento y la detección de errores.

3.2.1.- ACCESO A REGISTROS DEL SISTEMA

Se accede a los registros del sistema mediante el uso de direcciones de memoria que se le asigna a cada uno, los valores de estas direcciones se dan a partir de una dirección base (CC000h) que corresponde a la tarjeta Constellation-E [11]. Los registros de escritura utilizados durante el proceso de una imagen tienen un ancho de 1 byte (ver tabla 3.1).

Dirección	Función
Base+0000	Escritura de los Píxeles de la Imagen o el SE
Base+0001	Escritura en el Registro de Banderas de Entrada
Base+0002	Escritura del Ancho de la Imagen
Base+0003	Escritura de la Altura de la Imagen

TABLA 3.1.- FUNCIONES DE ESCRITURA DE LA INTERFAZ ISA

Los registros que puede leer el CP realizan diferentes funciones, siendo ventajosos en el procesamiento de una imagen o en la verificación del funcionamiento del sistema (ver tabla 3.2). El acceso a los registros de verificación se dal cuando el CP realiza pruebas de funcionamiento paso a paso, en cambio, los registros de Resultado y el Registro de Banderas de salida son usados en el procesamiento de una imagen.

Dirección	Función	Finalidad
Base+0000	Lectura del Resultado de la Dilación	Procesamiento
Base+0001	Lectura del Resultado de la Erosión	Procesamiento
Base+0002	Lectura del Contador de Columnas del Resultado	Verificación
Base+0003	Lectura del Contador de Filas del Resultado	Verificación
Base+0004	Lectura del Registro de Banderas de salida	Procesamiento
Base+0005	Lectura del Registro de Datos de Entrada	Verificación
Base+0006	Lectura del Contador de Columnas de Ingreso de Datos	Verificación
Base+0007	Lectura del Contador de Filas de Ingreso de Datos	Verificación
Base+0008	Lectura del Registro de Ancho de la Imagen Ingresado	Verificación
Base+0009	Lectura del Registro de la Altura de la Imagen Ingresado	Verificación

TABLA 3.2.- FUNCIONES DE LECTURA DE LA INTERFAZ ISA

3.2.2.- INTERFAZ USUARIO

La interfaz usuario presenta un menú (ver tabla 3.3) que consta de 7 opciones que permiten escoger una tarea a realizar por el CP.

Opción	Función a Realizar
1	Convertir BMP a hex
2	Ingresar Elemento Estructural
3	Prueba Funcional del Sistema (imágenes pequeñas)
4	Pruebas de Rendimiento del Sistema
5	Filtrado de la Imagen
6	Convertir hex a BMP
7	Salir

TABLA 3.3.- OPCIONES DEL MENÚ DE LA INTERFAZ USUARIO

La opción 1 convierte una imagen en escala de grises con formato BMP (ver anexo B.10) a un archivo binario con extensión 'hex' que pueda ser entendido por el sistema; este archivo (*A.hex*) solo contiene la información de tamaño de imagen y los valores de los píxeles que la conforman. La opción 2 permite que el usuario ingrese por teclado la información del SE, la cual se almacena en un archivo con una extensión 'ejm' (*SE.ejm*).

Las opciones 1 y 2 deben ejecutarse antes que las opciones 4 y 5, debido a que estas últimas requieren la información de una imagen y un SE; en cambio la opción 3 procesa una pequeña imagen (tamaño menor a 12x12 píxeles) que es ingresada por teclado y almacenada en un archivo *A* con extensión 'ejm'.

La prueba funcional del sistema (opción 3) verifica los pasos dados en el proceso de la imagen almacenada en *A.ejm;* en todo momento se imprime en pantalla el estado de las banderas de control y los contadores del sistema,

luego del proceso se imprimen los resultados de las operaciones de dilación y erosión. Por otro lado en las pruebas de rendimiento (opción 4) se obtienen los tiempos de procesamiento promedio de un número de muestras que es escogido por el usuario, estos tiempos son registrados en un archivo reporte con extensión 'txt'.

El filtrado de la imagen (opción 5) obtiene los resultados tanto de dilación como de erosión en dos archivos resultado, DILA.HEX y ERO.HEX; con la opción 6 se convierten estos archivos en imágenes con formato BMP en escala de grises de 8 bits, las cuales pueden visualizarse desde cualquier editor gráfico.

3.3.- PRUEBA FUNCIONAL DE LA ARQUITECTURA

Con la finalidad de analizar el correcto funcionamiento de los filtros implementados, se realizan pruebas en imágenes que tienen como dimensión máxima 12x12 píxeles y mínima 5x5 píxeles, debido a que con un tamaño mayor sería tedioso el análisis de los resultados en un entorno visual.

Por ejemplo, la dilación de una imagen con un SE tipo cruz (ver figura 3.2) permite un efecto de expansión de los tonos claros hacia 4 direcciones: arriba, abajo, izquierda, derecha. Esto ocasiona un efecto similar con los tonos oscuros en el caso de la erosión.

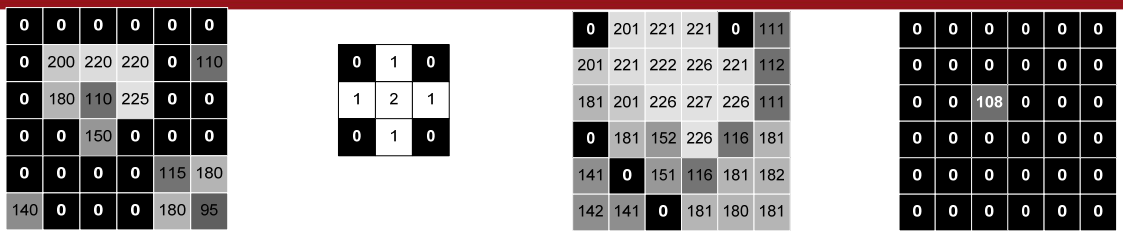


FIGURA 3.2.- RESULTADO DE OPERACIONES MORFOLÓGICAS (A) IMAGEN A 6x6, (B) SE 3x3, (C) IMAGEN DILATADA, (D) IMAGEN EROSIONADA

Con estos tamaños de imágenes es posible verificar los resultados de los diferentes procesos. En el caso de la dilación se puede apreciar el enriquecimiento de píxeles claros, mientras que para la erosión se ve un empobrecimiento de los mismos con respecto al fondo de la imagen. La interfaz usuario solo permite ver los valores numéricos en una estructura matricial, debido a que está diseñada para ser observada en entorno tipo texto. Los resultados coinciden con los obtenidos por los algoritmos diseñados en Matlab.

3.4.- TIEMPOS DE PROCESAMIENTO

3.4.1.- MÁXIMA VELOCIDAD DE PROCESAMIENTO OBTENIBLE

Dado que el sistema realiza el proceso de filtrado en paralelo con la entrada/salida de datos, la máxima velocidad de procesamiento que es posible obtener del sistema depende de la velocidad de la arquitectura y la velocidad de transferencia de datos de la interfaz E/S. El tiempo de procesamiento teórico del sistema (ver tabla 3.4) se obtiene de las simulaciones en Max+PlusII (ver figura 3.3) utilizando la máxima frecuencia de reloj alcanzable que fue de 58.47

MHz, estas simulaciones no contemplan los tiempos de acceso al Bus ISA [19] debido a que estos tienen una demora considerable en comparación al sistema diseñado.

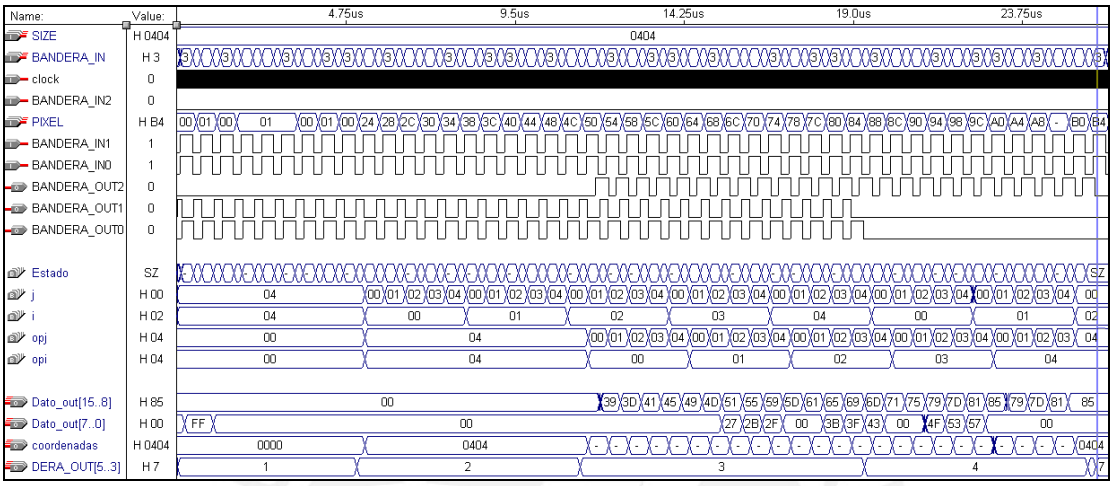


FIGURA 3.3.- DIAGRAMA DE TIEMPOS PARA UNA IMAGEN DE 5X5 PÍXELES Y UN SE DE 3X3 PÍXELES

Dimensión	Tiempo de Procesamiento (us)	Cuadros por Segundo (fps)
8x8	58.8	17006.80
16x16	220	4545.45
32x32	866.5	1154.06
64x64	3451	289.77
128x128	13789	72.52
256x256	55140	18.13

TABLA 3.4.- TIEMPOS TEÓRICOS DE PROCESAMIENTO DEL SISTEMA PARA DIFERENTES TAMAÑOS DE IMAGEN

Se realizó una comparación cualitativa del Sistema diseñado con otras arquitecturas [21] [22] para imágenes con tamaño 256x256 píxeles (ver tabla 3.5). Se puede apreciar que el sistema procesa más cuadros por segundo que

el procesador DSP TMS320C40 de Texas Instruments [23] pero menos que el FPGA XC4036EX-2 de Xilinx [24].

Producto y Compañia	Frecuencia de Reloj (MHz)	Tiempo Total de Procesamiento (ms)	Cuadros por segundo (fps)
TMS320C40 Texas Instruments	133	170	5.88
FLEX10K100E Sistema Diseñado	58.47	55.14	18.13
XC4036EX-2 Xilinx	75	11.36	88

TABLA 3.5.- COMPARACIÓN DEL FILTRO MORFOLÓGICO DISEÑADO CON OTRAS ARQUITECTURAS PARA IMÁGENES DE 256x256 PÍXELES

3.4.2.- TIEMPOS DE PROCESAMIENTO EXPERIMENTALES

En la implementación realizada se pudo comprobar que la velocidad de acceso al Bus ISA establece los tiempos de procesamiento, los cuales se obtuvieron luego de realizar pruebas de Rendimiento del Sistema con el CP.

Se tomaron los tiempos promedio del sistema para diferentes tamaños de imagen, evaluados sobre una población de 1000 muestras para cada caso. Asimismo se evaluaron los tiempos de duración de proceso para cada operación morfológica con los algoritmos implementados en Matlab, pero sobre una población de 100 muestras (ver tabla 3.6).

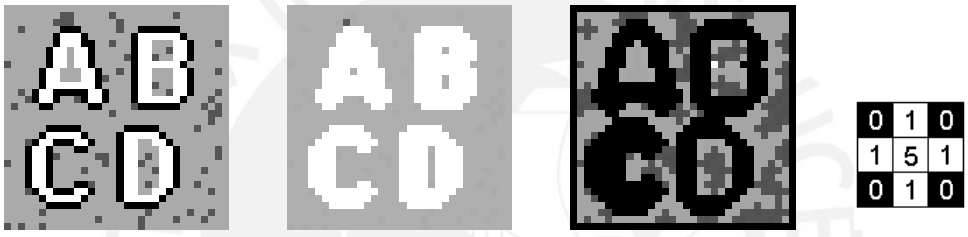
Dimensión	Medio	Tiempo Total de Procesamiento (ms)	
		Dilación	Erosión
8x8	Algoritmo en MATLAB	19.95	23.85
	Sistema Implementado	1.84	
16x16	Algoritmo en MATLAB	71.8	97.75
	Sistema Implementado	2.53	
32x32	Algoritmo en MATLAB	292.95	385.15
	Sistema Implementado	5.91	
64x64	Algoritmo en MATLAB	1177.4	1542.85
	Sistema Implementado	18.26	
128x128	Algoritmo en MATLAB	4724.15	6172.7
	Sistema Implementado	68.98	
256x256	Algoritmo en MATLAB	18985.95	24836.05
	Sistema Implementado	274.34	

TABLA 3.6.- COMPARACIÓN DE LOS TIEMPOS DE PROCESAMIENTO EXPERIMENTALES ENTRE EL SISTEMA IMPLEMENTADO Y MATLAB

Puede verse que para distintos tamaños de imagen los algoritmos de Dilación y Erosión en Matlab toman diferentes tiempos, la Dilación se realiza un poco más rápido que la Erosión; las pruebas de los algoritmos en Matlab fueron realizadas en una PC Pentium IV de 1.8 GHZ y 256 Mbytes de RAM. En el caso del sistema implementado, tanto el proceso de Dilación y Erosión toman el mismo tiempo de procesamiento pues la Arquitectura realiza el filtrado en paralelo. Analizando los datos se ve que el Sistema Implementado en el FPGA procesa las imágenes mucho más rápido que los algoritmos en Matlab.

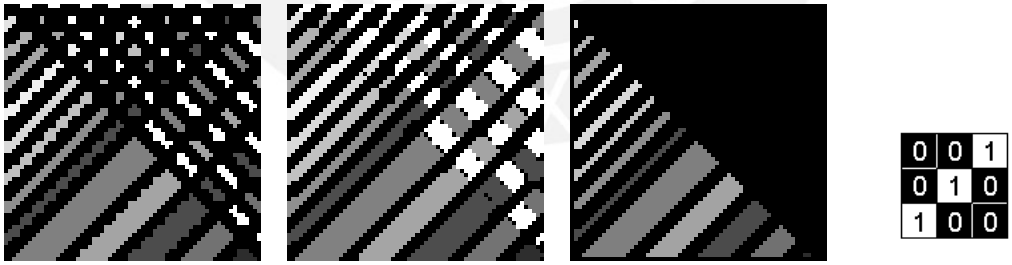
3.5.- RESULTADOS OBTENIDOS

Las imágenes con las cuales se probó el sistema fueron letras.bmp, franjas.bmp, ws.BMP, snoopy.bmp y texto.bmp con dimensiones de 32x32, 64x64, 128x128, 255x222 y 256x256 píxeles respectivamente. Las pruebas dieron un error nulo al ser comparados con los resultados obtenidos por Matlab. Las figuras 3.4, 3.5, 3.6, 3.7 y 3.8 muestran las imágenes junto con el SE con las que fueron filtradas y los resultados de Dilación y Erosión.



(A) (B) (C) (D)

FIGURA 3.4.- (A) IMAGEN ORIGINAL DE 32X32 PIXELES, (B) IMAGEN DILATADA, (C) IMAGEN EROSIONADA, (D) SE CON FORMA DE CRUZ



(A) (B) (C) (D)

FIGURA 3.5. - (A) IMAGEN ORIGINAL DE 64X64 PIXELES, (B) IMAGEN DILATADA, (C) IMAGEN EROSIONADA, (D) SE CON FORMA DIAGONAL



1	1	1
1	1	1
1	1	1

(A) (B) (C) (D)

FIGURA 3.6.- (A) IMAGEN ORIGINAL DE 128X128 PIXELES, (B) IMAGEN DILATADA, (C) IMAGEN EROSIONADA, (D) SE CON FORMA CUADRADA

(A)

1	1	1
1	1	1
1	1	1

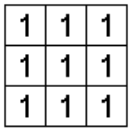
(B)

(C)

(D)

FIGURA 3.7.- (A) IMAGEN ORIGINAL DE 255X222 PIXELES, (B) SE CON FORMA CUADRADA, (C) IMAGEN DILATADA, (D) IMAGEN EROSIONADA

(A)



(B)

(C)

(D)

FIGURA 3.8.- (A) IMAGEN ORIGINAL DE 256X256 PIXELES, (B) SE CON FORMA CUADRADA, (C) IMAGEN DILATADA, (D) IMAGEN EROSIONADA

4. CONCLUSIONES

- La implementación de la arquitectura diseñada mostró una utilización eficiente del espacio en el FLEX10K100EQC240-3 de Altera, que fue del 42%, esto deja suficiente espacio para la implementación de otros módulos de post-procesamiento. No fue necesario el uso de memorias externas porque se aprovecho la memoria interna del FPGA necesaria. Por otro lado, el diseño supero la restricción de tener como mínima frecuencia de reloj 40 Mhz en la tarjeta de desarrollo.

- Los bloques de filtrado unidimensional ocupan el mayor número de celdas lógicas. Modificar estos bloques para un SE de 5x5 píxeles requeriría un consumo aproximadamente 8 veces mayor, esto no sería recomendable en un FLEX10K100EQC240 de Altera desde el punto de vista de ocupación y costos. Lo más apropiado sería la implementación de un filtro morfológico adicional como etapa de post-procesamiento, así aprovechando la propiedad de composición de las operaciones morfológicas se obtendrían los mismos resultados, y además se podrían lograr otras operaciones tales como apertura y cierre.

- La arquitectura pipeline usada brinda una solución a las limitaciones de transferencia de datos inherentes en otras arquitecturas que utilizan arreglos de procesamiento separados. Además se aprecia un beneficio en aumentar las velocidades de procesamiento si se procesan varias imágenes como una larga trama de datos, lo cual permite aplicaciones en tiempo real.

- Se comprueba la gran eficiencia y precisión que brindan las herramientas de simulación y síntesis de los diseños hechos con VHDL en diferentes FPGAs. A pesar que el desarrollo de algoritmos en FPGA para procesamiento de imágenes es tedioso, debido a la complejidad de la descripción.

- El diseño presentado en este trabajo es capaz de realizar otros tipos de filtrado de imágenes que involucren operaciones sobre vecindades de píxeles. De hecho, teniendo las cadenas de registros que facilitan los datos de una vecindad, y el control mediante contadores de fila y columna, una variedad de aplicaciones son posibles tan solo con variar los bloques aritméticos, lo cual puede hacer que realicen distintas operaciones de filtrado de imágenes. Por ejemplo: se puede implementar la convolución si se reemplazan los sumadores morfológicos por multiplicadores y los comparadores por un circuito que promedie.

- Realizar implementaciones de operaciones morfológicas mas complejas basadas en la dilación y la erosión, requeriría FPGAs con un número mayor a 5000 celdas lógicas, para que puedan caber varios filtros morfológicos presentados en este trabajo junto con las debidas interfaces. Este tipo de implementaciones se podría realizar sobre los FPGA de la familia Stratix que son de mayor capacidad.

- El diseño toma ventaja del paralelismo que es posible lograr en los FPGAs, por ello se pueden realizar las operaciones de dilación y erosión a la vez; además se tiene la ventaja de obtener, mediante la resta de estas operaciones, el gradiente morfológico que muestra los bordes de las figuras presentes en ella.

- Configurar la arquitectura para procesar imágenes de mayor tamaño no requiere un aumento considerable en el número de celdas lógicas, sin embargo los tiempos de procesamiento crecen en proporción al tamaño de la imagen, lo cual no hace recomendable el uso de esta arquitectura en imágenes de gran tamaño.

- El diseño presentado es portátil a implementaciones en diferentes dispositivos de transferencia de datos tales como PCI, debido a que la unidad de control posee estados de espera y sincroniza la transferencia en base a banderas de control.

- Las pruebas obtenidas de la implementación en *hardware* mostraron una alta confiabilidad porque no mostraron ninguna diferencia con los resultados obtenidos en MATLAB en todas las pruebas realizadas.

- Se ha logrado, en la Pontificia Universidad Católica del Perú, extender el estudio de la línea de investigación de procesamiento digital de imágenes, a tratamiento morfológico de imágenes en escala de grises, siendo los resultados obtenidos adecuados dentro de nuestro contexto tecnológico. El conocimiento y la experiencia ganados serán de mucha ayuda para completar futuros diseños.

5. RECOMENDACIONES

- Para algunas aplicaciones con imágenes de mayor tamaño (512x512) los SE de 3x3 píxeles dejan de ser de gran utilidad, es preferible utilizar tamaños de 5x5 y 7x7 píxeles; el problema en esto radica en las limitaciones de espacio y memoria, se tendrían que usar FPGAs con mayor capacidad que el FLEX10K100E. Por ello se considera que una extensión de este trabajo sería la implementación de un filtro morfológico configurado con un elemento estructural de mayores dimensiones.

- Debido que la arquitectura es modular se pueden implementar otros tipos de filtrado morfológico más complejos mediante la concatenación de varios de los filtros morfológicos básicos junto con operadores aritméticos. Las interconexiones entre filtros se hacen sencillas porque bastaría que se adapten las banderas de control de transferencia entre ellos, pero se recomienda agregar una unidad de control general que administre la transferencia de resultados de los diferentes filtros y operadores aritméticos.

- Con el uso de 3 filtros morfológicos en paralelo se podría lograr tratamiento morfológico de imágenes RGB, de manera que cada filtro procese un color (rojo, verde o azul) por separado.

- Los bloques comparadores descritos estructuralmente alcanzan mayores frecuencias de reloj pero consumen mayor cantidad de celdas lógicas. Se recomienda el uso de estos comparadores sobre todo con palabras de gran ancho, en aplicaciones en donde sea necesario alcanzar mayores frecuencias de reloj o no sea crítico el ahorro de celdas lógicas.

REFERENCIAS

[1] Pierre Soille, “Morphological Images Analysis”. Springer-Verlag Berlin Heidelberg 1999

[2] Milan Sonka, Vaclav Hlavac and Roger Boyle, “Image Processing, Analysis and Machine Vision”, PWS Publishing, segunda edicion, 1999

[3] Joseph H. Bosworth and Scott T. Acton, “Morphological scale-space in image processing”, Department of Electrical Engineering, University of Virginia, Charlottesville, 2002

[4] Yoshimoto N., Ernesto Akio, “Implementación de un Algoritmo para Tratamiento Morfológico de Imágenes Digitales en un CPLD de Altera”, Tesis de Licenciatura, Pontificia Universidad Católica Del Perú, Facultad de Ciencias e Ingenieria, Lima-Perú, 2002

[5] Stephen Brown and Jonathan Rose, “Architecture of FPGAs and CPLDs: A Tutorial”, *Department of Electrical and Computer Engineering University of Toronto, 2000*

[6] Hussain Zahid, “Digital Image Processing: Practical applications of parallel processing techniques”, Ellis Horwood Limited, 1991

[7] A. L. Abbott, R. M. Haralick, and X. Zhuang, “Pipeline Architectures for Morphologic Image Analysis”, Machine Vision and Applications 1 (1); 23-40

[8] Altera, “Flex 10k embedded programmable logic family data sheet”, mayo del 2000, version 4.02

[9] Altera, “Stratix FPGA family data sheet”, Diciembre del 2002, versión 3.0

[10] Altera, “ACEX1K Programable Logic Device Family data sheet”, Mayo del 2003, versión 3.4

[11] Nova Engineering, Inc., “Constellation E development system user’s manual”, Cincinnati, Ohio 45246, revision 1.0.

[12] Hunt, Brian R., “A guide to MATLAB : for beginners and experienced users” *Cambridge, UK. ; New York : Cambridge University Press*, 2001

[13] Adit Tarmaster, Peter M. Athanas, and A. Lynn Abbott, "Accelerating Image Filters Using a Custom Computing Machine", The Bradley Department of Electrical Engineering Virginia Polytechnic Institute and State University Blacksburg, Virginia, 1999

[14] Carlos Coelho, Nuno Roma and Leonel Sousa, “Pipeline Architectures for Computing 2-D Image Moments”, Dept. of Electrical Engineering, INESC

[15] Actel Corporation, “Actel HDL Coding, Style guide”, 2000

[16] Stephen Brown and Zvonko Vranesic, “Fundamentals of Digital Logic with VHDL Design”, McGraw-Hill 2000

[17] Altera, “Altera digital library”. marzo 2002.

[18] U. Meyer – Baese, “Digital Signal Processing with Field Programable Gate Arrays”. Springer-Verlag Berlin Heidelberg 2001

[19] Intel Datasheet Addendum. 82371AB (PIIX4) PCI ISA IDE Xcelerator Timing Specifications, Setiembre 1997

[20] Amir M. Tahmasebi - "Matlab Tutorial Image Processing Toolbox", 2002

[21] Pujol, F.A.[1]; García Chamizo, J.M.[1]; Ledesma, B.[1]; Fuster, A.[1]; Pujol, M[2]; “DSP Implementations of Morphological Filtering: Application to Path Planning Problems”, [1]Departamento de Tecnología Informática y Computación, [2]Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, 2000

[22] D. Crookes, K. Bendrik, A. Bouridane, K. Alotabi and A. Bendrik, “Design and implementation of a high level programming environment for FPGA-based image processing”, IEE Proc.-Vis. Image Signal Process., Vol 147, No. 4, Agosto 2000

[23] Texas Instruments, TMS320C4x. User’s Guide, 1996

[24] Xilinx, “XC4000XLA/XV Field Programable Gate Arrays”, Octubre 99, version 1.3

ANEXOS

ANEXO A: CÓDIGO DE DESCRIPCIÓN EN HARDWARE DE LOS BLOQUES DEL SISTEMA

A.1. UNIDAD DE CONTROL Y ALMACENAMIENTO DE DATOS

CONTROL_DATOS.VHD

```
library ieee;
use ieee.std_logic_1164.all;

USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity CONTROL_DATOS is
       generic (      ancho_palabra  : natural := 8;
                      tamano : natural := 8;
                      total_filas : natural :=8;
                      n_filas : natural := 2
                );
       port(   Dato_in : in    std_logic_vector(ancho_palabra - 1 downto 0);

                width : in std_logic_vector(tamano-1 downto 0);
                heigth :  in std_logic_vector(total_filas-1 downto 0);
-- Señales de control del Chip
                clock, rst     : in std_logic;
-- Banderas de entrada y salida--
                envio_dato     : in std_logic;
                prep_para_recib : out std_logic;
                dato_recibido : out std_logic;

                R_listo : out std_logic;

                R_read : in std_logic;
-------------------------------------------------
-- Señales de salida y Control interno
       SF0_out, SF1_out, SF2_out, SF3_out: out std_logic_vector(ancho_palabra-1 downto 0);

       SE1_out, SE2_out, SE3_out : out std_logic_vector(ancho_palabra*3 - 1 downto 0);

-- Señales de control de los Bloques de Ordenamiento
                hab_col, hab_fila, last_row : out std_logic;
                inicio_fila, final_fila : out std_logic;

-- Relacion de señales para VISUALIZACION EXTERNA
                j_out : out std_logic_vector(tamano-1 downto 0);
                i_out : out std_logic_vector(total_filas-1 downto 0);
                opj_out : out std_logic_vector(tamano-1 downto 0);
                opi_out : out std_logic_vector(total_filas-1 downto 0);
                estado_out : out std_logic_vector(2 downto 0)
                );
end CONTROL_DATOS;

architecture arqui of CONTROL_DATOS is

-- Defino tipos para los estados
       type estado_tipo is (SA, S0, S1, S2, S3, S4, S5, S6, S7, SZ);
       signal estado : estado_tipo;

       signal unos, zeros     : std_logic_vector(ancho_palabra - 1 downto 0);

       signal SF0, SF1, SF2, SF3 : std_logic_vector(ancho_palabra - 1 downto 0);

-- señal de control de escritura
```

```
        signal w,wt : std_logic_vector(3 downto 0);

        signal Escribe_SE_r, Escribe_SE : std_logic;
        signal Escribe_MEM_r, Escribe_MEM : std_logic;

-- elemento estructual 3x3
        signal SE11, SE12, SE13 : std_logic_vector(ancho_palabra - 1 downto 0);
        signal SE21, SE22, SE23 : std_logic_vector(ancho_palabra - 1 downto 0);
        signal SE31, SE32, SE33 : std_logic_vector(ancho_palabra - 1 downto 0);

-- señales de control de los registros del SE
        signal habil_SE, habil_SEt : std_logic_vector(8 downto 0);

        --Señal de control del ingreso de los pixels del Elemento Estructural
        signal n_pixel_SE : std_logic_vector(3 downto 0);

-- Defino indices de fila y columna
        signal j, opj : std_logic_vector(tamano-1 downto 0);
        signal adress : std_logic_vector(tamano-1 downto 0);
        signal i, opi: std_logic_vector(total_filas-1 downto 0);
        signal im: std_logic_vector(n_filas-1 downto 0);

------------------------------------------------
        --SEÑALES DE CONTROL DE CONTADORES    TEMPORALES

        signal cl_j_t, cl_j, cl_opj_t, cl_opj : std_logic;
        signal cl_i_t, cl_i, cl_opi_t, cl_opi : std_logic;

  --habilitadores de cuenta y señales de inicio
        signal hab_jt, set_jt : std_logic;
        signal hab_it, set_it : std_logic;
        signal hab_SEt, set_SEt : std_logic;
        signal hab_opjt, set_opjt : std_logic;
        signal hab_opit, set_opit : std_logic;
  --indicadores de valor maximo (seria interesante agrupar estas señales en un vector)
        signal max_jt, max_imt: std_logic;
        signal max_it, max_SEt, max_opjt, max_opit: std_logic;

        signal fin_ingresot, fin_procesot: std_logic;
  --señal que controla la conmutacion de filas hacia los operadores
        signal cambia_opfilat : std_logic;

--SEÑALES DE CONTROL DE CONTADORES
  --habilitadores de cuenta y señales de inicio
        signal hab_j, set_j : std_logic;
        signal hab_i, set_i : std_logic;
        signal hab_SE, set_SE : std_logic;
        signal hab_opj, set_opj : std_logic;
        signal hab_opi, set_opi : std_logic;
  --indicadores de valor maximo (seria interesante agrupar estas señales en un vector)
        signal max_j, max_im: std_logic;
        signal max_i, max_SE, max_opj, max_opi: std_logic;
        signal fin_ingreso, fin_proceso: std_logic;
  --señal que controla la conmutacion de filas hacia los operadores
        signal cambia_opfila : std_logic;
-----------------------------------------------------------------

  --Banderas de cambio de estado
        signal c_0a1t, c_1a0t, c_1a2t : std_logic; --para carga del SE
        signal c_2a3t, c_3a2t, c_3a4t : std_logic; --para carga de las 2 primeras filas
        signal c_4a5t, c_5a4t, c_5a6t : std_logic; --para carga y procesamiento
        signal c_6a7t, c_7a6t, c_7aZt : std_logic; --procesamiento ultimas filas

        signal c_0a1, c_1a0, c_1a2 : std_logic; --para carga del SE
        signal c_2a3, c_3a2, c_3a4 : std_logic; --para carga de las 2 primeras filas
        signal c_4a5, c_5a4, c_5a6 : std_logic; --para carga y procesamiento
        signal c_6a7, c_7a6, c_7aZ : std_logic; --procesamiento ultimas filas

  --Señales del temporizador para la señal de control del Resultado
        signal temporiza : std_logic_vector(3 downto 0);
        signal hab_temporiza, dato_procesado : std_logic;
```

```
        signal hab_temporizat, dato_procesadot : std_logic;

        signal cl_temp, cl_temp_t : std_logic;

  --Bandera que muestra el estado a la salida
        signal estado_out_t : std_logic_vector(2 downto 0);

  --Señales de Sincronizacion de las SEÑALES DE CONTROL con los DATOS de las memorias
        signal hab_col1, hab_col2, hab_col3 : std_logic;
        signal i_col1, i_col2, i_col3, i_col4 : std_logic;
        signal u_col1, u_col2, u_col3, u_col4 : std_logic;

        signal c_fil1, c_fil2, c_fil3 : std_logic;
        signal l_r1, l_r2, l_r3, l_r4 : std_logic;
begin
-- Relacion de SEÑALES DE CONTROL de salida hacia los otros bloques

  -- BLOQUE ORDENA COLUMNAS:
        -- indica la habilitacion de los bloques de ordenamiento de Operadores
        hab_col <= hab_col3;
-- los momentos apropiados

        -- indica la correccion del efecto de borde en la primera columna IZQUIERDA
        inicio_fila <= i_col4;
        -- indica la correccion del efecto de borde en la ultima columna DERECHA
        final_fila <= u_col4;

  -- BLOQUE ORDENA FILAS:
        -- indica la habilitacion de cambio de filas
        hab_fila <= c_fil3;
        -- indica la correccion del efecto de borde de la ultima fila
        last_row <= l_r4;

-- Relacion de señales para VISUALIZACION EXTERNA
        j_out <= j;
        i_out <= i;

        opj_out <= opj;
        opi_out <= opi;

--SALIDAS DE LAS MEMORIAS
        SF0_out <= SF0;
        SF1_out <= SF1;
        SF2_out <= SF2;
        SF3_out <= SF3;

        --salidas del elemento estrutural 3x3
        SE1_out(ancho_palabra*3-1 downto ancho_palabra*2) <= SE11; --fila superior
        SE1_out(ancho_palabra*2-1 downto ancho_palabra) <= SE12;
        SE1_out(ancho_palabra-1 downto 0) <= SE13;

        SE2_out(ancho_palabra*3-1 downto ancho_palabra*2) <= SE21; --fila media
        SE2_out(ancho_palabra*2-1 downto ancho_palabra) <= SE22;
        SE2_out(ancho_palabra-1 downto 0) <= SE23;

        SE3_out(ancho_palabra*3-1 downto ancho_palabra*2) <= SE31; --fila inferior
        SE3_out(ancho_palabra*2-1 downto ancho_palabra) <= SE32;
        SE3_out(ancho_palabra-1 downto 0) <= SE33;

--- Generic para obtener Vectores de 0s y 1s ----
gen1:   for i in 0 to ancho_palabra-1 generate
                    zeros(i) <= '0';
                    unos(i) <= '1';
              end generate;
-------------------------------------------------
```

```
-- ********************************************************************************
-- *** BLOQUE ESTRUCTURAL DE MEMORIAS DE ALMACENAMIENTO TEMPORAL DE FILAS ****
-- ********************************************************************************

fila0 : lpm_ram_dq
      GENERIC MAP (    lpm_widthad           => tamano,
                              lpm_outdata           => "REGISTERED",
                              lpm_indata            => "REGISTERED",
                              lpm_address_control   => "REGISTERED",
                              lpm_width             => ancho_palabra)
      PORT MAP (data => Dato_in, address => adress,
               we => w(0), inclock => clock, outclock => clock, q => SF0);

fila1 :  lpm_ram_dq
      GENERIC MAP (    lpm_widthad           => tamano,
                              lpm_outdata           => "REGISTERED",
                              lpm_indata            => "REGISTERED",
                              lpm_address_control   => "REGISTERED",
                              lpm_width             => ancho_palabra)
      PORT MAP (data => Dato_in, address => adress,
               we => w(1), inclock => clock, outclock => clock, q => SF1);

fila2 : lpm_ram_dq
      GENERIC MAP (    lpm_widthad           => tamano,
                              lpm_outdata           => "REGISTERED",--un
                              lpm_indata            => "REGISTERED",
                              lpm_address_control   => "REGISTERED",--un
                              lpm_width             => ancho_palabra)
      PORT MAP (data => Dato_in, address => adress,
               we => w(2), inclock => clock, outclock => clock, q => SF2);

fila3 :  lpm_ram_dq
      GENERIC MAP (    lpm_widthad           => tamano,
                              lpm_outdata           => "REGISTERED",
                              lpm_indata            => "REGISTERED",
                              lpm_address_control   => "REGISTERED",
                              lpm_width             => ancho_palabra)
      PORT MAP (data => Dato_in, address => adress,
               we => w(3), inclock => clock, outclock => clock, q => SF3);

--************************************
--******** Fin de Memorias ************
--************************************


-- ************************************************
-- **  BLOQUE ESTRUCTURAL DE CONTADORES DE CONTROL **
-- ****************** COUNTERS *********************
-- ************************************************

contador_SE: lpm_counter
   GENERIC MAP (LPM_WIDTH => 4)
        PORT MAP (     aset => set_SE,
                       clock => clock,
                       cnt_en => hab_SE,
                       q => n_pixel_SE);

contador_j: lpm_counter
   GENERIC MAP (LPM_WIDTH => tamano)
        PORT MAP (     data => width,
                       aload => set_j,
                       aclr => cl_j,
                       clock => clock,
                       cnt_en => hab_j,
                       q => j);

contador_opj: lpm_counter
   GENERIC MAP (LPM_WIDTH => tamano)
        PORT MAP (     data => width,
                       aload => set_opj,
                       aclr => cl_opj,
```

```
                        clock => clock,
                        cnt_en => hab_opj,
                        q => opj);

contador_opi: lpm_counter
   GENERIC MAP (LPM_WIDTH => total_filas)
        PORT MAP (      data => heigth,
                        aload => set_opi,
                        aclr => cl_opi,
                        clock => clock,
                        cnt_en => hab_opi,
                        q => opi);

contador_i: lpm_counter
   GENERIC MAP (LPM_WIDTH => total_filas)
        PORT MAP (      data => heigth,
                        aload => set_i,
                        aclr => cl_i,
                        clock => clock,
                        cnt_en => hab_i,
                        q => i);

contador_i_mem: lpm_counter
   GENERIC MAP (LPM_WIDTH => n_filas)
        PORT MAP (      aset => set_i,
                        clock => clock,
                        cnt_en => hab_i,
                        q => im);
--************************************
--******** Fin de COUNTERS ************
--************************************


--****************************************
--***** MAQUINA DE ESTADOS PRINCIPAL *****
--****************************************
Maquina: Process(clock, rst)
begin
        if rst='1' then
                estado <= SA;
        elsif clock'event and clock='1' then
                case estado is
                        when SA =>      estado <= S0;

                        when S0 =>
                                if c_0a1='1' then
                                        estado <= S1;
                                else estado <= S0;
                                end if;
                        when S1 =>
                                if c_1a2='1' then
                                        estado <= S2;
                                elsif c_1a0='1' then
                                        estado <= S0;
                                end if;

                        when S2 =>
                                if c_2a3='1' then
                                        estado <= S3;
                                else estado <= S2;
                                end if;
                        when S3 =>
                                if c_3a4='1' then
                                        estado <= S4;
                                elsif c_3a2='1' then
                                        estado <= S2;
                                end if;

                        when S4 =>
                                if c_4a5='1' then
                                        estado <= S5;
```

```
                                else estado <= S4;
                                end if;
                        when S5 =>
                                if c_5a6='1' then
                                        estado <= S6;
                                elsif c_5a4='1' then
                                        estado <= S4;
                                end if;

                        when S6 =>
                                if c_6a7='1' then
                                        estado <= S7;
                                else estado <= S6;
                                end if;
                        when S7 =>
                                if c_7aZ='1' then
                                        estado <= SZ;
                                elsif c_7a6='1' then
                                        estado <= S6;
                                end if;

                        when SZ =>
                        when others =>
                end case;
        end if;
end Process Maquina;

-- CONTROL DE HABILITADORES DE CAMBIO DE ESTADO
        --Estado de Carga del SE
        c_0a1t <='1' when envio_dato='1' else '0';
        c_1a0t <='1' when max_SE='0' and envio_dato='0' else '0';
        c_1a2t <='1' when max_SE='1' and envio_dato='0' else '0';

        --Estado de Carga de las 2 primeras filas
        c_2a3t <='1' when envio_dato='1' else '0';
        c_3a2t <='1' when max_im='0' and envio_dato='0' else '0';
        c_3a4t <='1' when max_im='1' and envio_dato='0' else '0';

        --Estado de Carga y Procesamiento
        c_4a5t <='1' when envio_dato='1' and ((max_opj='1' and max_opi='1' and max_im='1')
or  R_read='1') else '0';
        c_5a4t <='1' when fin_ingreso='0' and envio_dato='0' and R_read='0' and
dato_procesado='1' else '0';
        c_5a6t <='1' when fin_ingreso='1' and envio_dato='0' and R_read='0' and
dato_procesado='1' else '0';

        --Estado de Procesamiento de las 2 ultimas filas
        c_6a7t <='1' when R_read='1' else '0';
        c_7a6t <='1' when fin_proceso='0' and R_read='0' and dato_procesado='1' else '0';
        c_7aZt <='1' when fin_proceso='1' and R_read='0' and dato_procesado='1' else '0';

-- ************************************
-- *** Registro de Señales de Control ***
-- ************************************

Habilita: Process(clock)
 begin
        if clock'event and clock='1' then

-- CONTROL DE CAMBIOS DE ESTADO
                c_0a1 <= c_0a1t;
                c_1a0 <= c_1a0t;
                c_1a2 <= c_1a2t;

                c_2a3 <= c_2a3t;
                c_3a2 <= c_3a2t;
                c_3a4 <= c_3a4t;

                c_4a5 <= c_4a5t;
                c_5a4 <= c_5a4t;
                c_5a6 <= c_5a6t;
```

```
                c_6a7 <= c_6a7t;
                c_7a6 <= c_7a6t;
                c_7aZ <= c_7aZt;

------ *** CONTROL DE CONTADORES *** -----
        hab_j <= hab_jt;
        hab_i <= hab_it;
        hab_opj <= hab_opjt;
        hab_opi <= hab_opit;
        hab_SE <= hab_SEt;

cambia_opfila <= cambia_opfilat;

        --Banderas de Finalizacion de cuenta

        max_SE <= max_SEt;
        max_j <= max_jt;
        max_im <= max_imt;

        max_i <= max_it;
        max_opj <= max_opjt;
        max_opi <= max_opit;

         --señales de control para la maquina de estados
        fin_ingreso <= fin_ingresot;
        fin_proceso <= fin_procesot;

 --**señales de inicializacion**
        set_j  <= set_jt;
        set_opj<= set_opjt;
        set_opi<= set_opit;
        set_i  <= set_it;
        set_SE  <= set_SEt;
 -- ********************
        cl_j <= cl_j_t;
        cl_opj <= cl_opj_t;
        cl_i <= cl_i_t;
        cl_opi <= cl_opi_t;

 -- *********************

        hab_temporiza <= hab_temporizat;
        dato_procesado <= dato_procesadot;
        cl_temp <= cl_temp_t;

-- Señales de control de escritura
        w <= wt;
        habil_SE <= habil_SEt;
        adress <= j;

-- Señal de Salida que indica el estado en que la maquina se encuentra
        estado_out <= estado_out_t;

-- Sincronizacion de las señales con los Datos de las memorias
        hab_col1 <= hab_opj;
        hab_col2 <= hab_col1; --habilitador de columna
        hab_col3 <= hab_col2; --habilitador de columna

        i_col2 <= i_col1; --indice de inicio de fila
        i_col3 <= i_col2;
        i_col4 <= i_col3;

        u_col2 <= u_col1; --indice de fin de fila
        u_col3 <= u_col2;
        u_col4 <= u_col3;

        c_fil1 <= cambia_opfila;      --habilitador de cambio de fila
        c_fil2 <= c_fil1;
        c_fil3 <= c_fil2;
```

```
        l_r2 <= l_r1; --indice de fila final
        l_r3 <= l_r2;
        l_r4 <= l_r3;

        Escribe_SE <= Escribe_SE_r;
        Escribe_MEM <= Escribe_MEM_r;

        end if;
end Process Habilita;
--- ********************************************************** ---
-------------------------------------------------------------------

--*****************************************************************************
--*** Señales de comunicacion con la PC. Le dice a la PC que puede recibir datos ***
--***************** BANDERAS DE SALIDA PARA LA COMUNICACION ********************
--*****************************************************************************

        estado_out_t <=         "000" when estado=SA else
                                "001" when estado=S0 or estado=S1 else
                                "010" when estado=S2 or estado=S3 else
                                "011" when estado=S4 or estado=S5 else
                                "100" when estado=S6 or estado=S7 else
                                "101" when estado=SZ and dato_procesado='1' else
                                "111";

        with estado select
                prep_para_recib <=      '1'     when S0,
                                        '1'     when S2,
                                        '1'     when S4,
                                        '0'     when others;

        with estado select
                dato_recibido <=        '1' when S1,
                                        '1'     when S3,
                                        '1'     when S5,
                                        '0'     when others;

        R_listo <= dato_procesado; -- Indica que el Resultado esta listo, puede ser leido

        -- ***  TEMPORIZADOR DE OPERACIONES ***
         -- Tiene la funcion de contar el numero de ciclos clock que se demora el pipeline
         -- de los operadores matematicos para tener un resultadon listo en el registro de
         -- Salida de resultados
        temporizador: lpm_counter
           GENERIC MAP (LPM_WIDTH => 4)
                PORT MAP (      aclr => cl_temp,
                                clock => clock,
                                cnt_en => hab_temporiza,
                                q => temporiza);

                dato_procesadot <=      '1' when temporiza="1011" or temporiza="1010" else '0';
                -- Cuando el resultado esta listo, luego de 'n' pulsos de reloj entonces
                -- esta señal se activa y permite un cambio de estado

                hab_temporizat <='1' when (estado=S5 or estado=S7) and dato_procesadot='0'
else '0'; --le puse el t para ver si soluciona el
                -- SOLO se habilita la cuenta cuando la maquina se encuentra en un estado
ACTIVO

                cl_temp_t <= '1' when (c_4a5='1' and estado=S4) or (c_6a7='1' and estado=S6)
or (R_read='1' and estado=SZ) or estado=SA else '0';
        ------------------------------------------------------------------------
--*****************************************************************************

-- ************ CONTROL DE CONTADORES *************
  -- ** COUNTERS_HAB **
        hab_jt <=       '1' when estado=S2 and c_2a3='1' else
                        '1' when estado=S4 and c_4a5='1' else
                        '1' when estado=S6 and c_6a7='1' else
                        '0';
        hab_it <='1' when max_jt='1' and hab_jt='1' else '0';
```

```
       hab_opjt <=    '1' when estado=S4 and c_4a5='1' else
                      '1' when estado=S6 and c_6a7='1' else
                      '0';

       hab_opit <=    '1' when max_opjt='1' and hab_opjt='1' else '0';

       hab_SEt <='1' when estado=S0 and c_0a1='1' else '0';

 --**señales de inicializacion  ** COUNTERS_SET**
       set_jt  <=     '1' when estado=SA else '0';
       set_opjt <=    '1' when estado=S3 and max_jt='1' else '0';
       set_opit <=    '1' when estado=S3 and max_jt='1' else '0';
       set_it <=      '1' when estado=SA else '0';
       set_SEt <=     '1' when estado=SA else '0';

  -- ***Señales de reinicializacion de los contadores de columnas
  -- ** COUNTERS_CLR **
       cl_j_t <= '1' when max_j='1' and hab_j='1' else '0';
       cl_opj_t <= '1' when max_opj='1' and hab_opj='1' else '0';
       cl_i_t <= '1' when max_i='1' and hab_i='1' else '0';
       cl_opi_t <= '1' when max_opi='1' and hab_opi='1' else '0';

 --señales de control para la maquina de estados
fin_ingresot <=       '1' when max_it='1' and max_jt='1' else '0';
fin_procesot <=       '1' when max_opit='1' and max_opjt='1' else '0';

cambia_opfilat <='1' when (estado=S4 or estado=S6) and hab_it='1' else '0';
l_r1 <= '1' when (i=zeros(total_filas-1 downto 1) & '1') and (estado_out_t="011" or
estado_out_t="100")  else '0';

-- ************************************************
-- ***************** COMPARADORES ******************
max_SEt <= '1' when n_pixel_SE="1000" else '0';
max_jt <= '1' when j=width else '0';
max_imt <= '1' when i(n_filas-1 downto 0)="10" else '0';
max_it <= '1' when i=heigth else '0';
max_opjt <= '1' when opj=width else '0';
max_opit <= '1' when opi=heigth else '0';

-- *** Señales de CONTROL importantes para los otros bloques en funcion de las internas ***
i_col1 <= '1' when opj=zeros(tamano-1 downto 0) else '0';
u_col1 <= max_opjt;

-- ************** Fin de  COMPARADORES ***************
-- ************************************************
-------------------------------------------------------

-- Señales de Habilitacion de escritura
       Escribe_SE_r <= '1' when estado=S1; --Esta señal ayudara a sincronizar la
habilitacion
       Escribe_MEM_r <= '1' when estado=S3 or estado=S5;

-- ********** Generacion de Escritura en la Memoria *********
       wt <=  "0001" when im(n_filas-1 downto 0)="00" and Escribe_MEM='1' else
              "0010" when im(n_filas-1 downto 0)="01" and Escribe_MEM='1' else
              "0100" when im(n_filas-1 downto 0)="10" and Escribe_MEM='1' else
              "1000" when im(n_filas-1 downto 0)="11" and Escribe_MEM='1' else
              "0000";

-- **** Generacion de Escritura en los Registros del SE ****
       habil_SEt     <=     "000000001" when Escribe_SE='1' and n_pixel_SE="0000" else
                            "000000010" when Escribe_SE='1' and n_pixel_SE="0001" else
                            "000000100" when Escribe_SE='1' and n_pixel_SE="0010" else
                            "000001000" when Escribe_SE='1' and n_pixel_SE="0011" else
                            "000010000" when Escribe_SE='1' and n_pixel_SE="0100" else
                            "000100000" when Escribe_SE='1' and n_pixel_SE="0101" else
                            "001000000" when Escribe_SE='1' and n_pixel_SE="0110" else
                            "010000000" when Escribe_SE='1' and n_pixel_SE="0111" else
                            "100000000" when Escribe_SE='1' and n_pixel_SE="1000" else
                            "000000000";
```

```
-- ****** REGISTROS DEL ELEMENTO ESTRUCTURAL *******
registroSE: Process(clock)
begin
--fila1
        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(0)='1' THEN
                        SE11 <= Dato_in;
                end if;
        END IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(1)='1' THEN
                        SE12 <= Dato_in;
                end if;
        E ND IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(2)='1' THEN
                        SE13 <= Dato_in;
                end if;
        END IF;

--fila2
        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(3)='1' THEN
                        SE21 <= Dato_in;
                end if;
        END IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(4)='1' THEN
                        SE22 <= Dato_in;
                end if;
        END IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(5)='1' THEN
                        SE23 <= Dato_in;
                end if;
        END IF;

--fila3
        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(6)='1' THEN
                        SE31 <= Dato_in;
                end if;
        END IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(7)='1' THEN
                        SE32 <= Dato_in;
                end if;
        END IF;

        IF (clock'EVENT and clock='1') THEN
                IF habil_SE(8)='1' THEN
                        SE33 <= Dato_in;
                end if;
        END IF;

end Process registroSE;

end arqui;
```

A.2. BLOQUE DE ORDENAMIENTO DE FILAS DE LA RAM

ORDENA_FILAS.VHD

```
library ieee;
use ieee.std_logic_1164.all;

entity ordena_filas is
	generic (	ancho_palabra  : natural := 8;
			n_filas : natural := 2);
			--Cuatro memorias, una por cada fila temporal
	port(	M0, M1, M2, M3 : in   std_logic_vector(ancho_palabra - 1 downto 0);
		clock, rst, hab_fila, low_row : in std_logic;
--------------
		F0, F1, F2 : out std_logic_vector(ancho_palabra - 1 downto 0)
		);
end ordena_filas;

architecture arqui of ordena_filas is
-- Defino tipos para los estados
	type estado_tipo is (SA, S0, S1, S2, S3);
	signal estado : estado_tipo;

	signal fila0, fila1, fila2, fila3 : std_logic_vector(ancho_palabra-1 downto 0);
	signal fila0t, fila1t, fila2t, fila3t : std_logic_vector(ancho_palabra-1 downto 0);
	signal F0t, F1t, F2t : std_logic_vector(ancho_palabra-1 downto 0);
	signal unos, zeros : std_logic_vector(ancho_palabra - 1 downto 0);
	signal asgn0, asgn1, asgn2, asgn3 : std_logic;
	signal asgn0t, asgn1t, asgn2t, asgn3t : std_logic;
begin

gen1:	for i in 0 to ancho_palabra-1 generate
		zeros(i) <= '0';
	end generate;

asgn0t <= '1' when low_row='1' and estado=S3 else '0';
asgn1t <= '1' when low_row='1' and estado=S0 else '0';
asgn2t <= '1' when low_row='1' and estado=S1 else '0';
asgn3t <= '1' when low_row='1' and estado=S2 else '0';

Sincroniza: Process(clock)
begin
  if clock'event and clock='1' then
	asgn0 <= asgn0t;
	asgn1 <= asgn1t;
	asgn2 <= asgn2t;
	asgn3 <= asgn3t;

	fila0 <= fila0t;
	fila1 <= fila1t;
	fila2 <= fila2t;
	fila3 <= fila3t;

	F0 <= F0t;
	F1 <= F1t;
	F2 <= F2t;

  end if;
end Process Sincroniza;

fila0t <=	zeros when asgn0='1' else M0;
fila1t <=	zeros when asgn1='1' else M1;
fila2t <=	zeros when asgn2='1' else M2;
fila3t <=	zeros when asgn3='1' else M3;

Maquina: Process(clock, rst)
begin
	if rst='1' then
		estado <= SA;
```

```
        elsif clock'event and clock='1' then
                case estado is
                        when SA =>
                                if hab_fila='1' then
                                        estado <= S1;
                                else
                                        estado <= SA;
                                end if;
                        when S0 =>
                                if hab_fila='1' then
                                        estado <= S1;
                                else
                                        estado <= S0;
                                end if;
                        when S1 =>
                                if hab_fila='1' then
                                        estado <= S2;
                                else
                                        estado <= S1;
                                end if;
                        when S2 =>
                                if hab_fila='1' then
                                        estado <= S3;
                                else
                                        estado <= S2;
                                end if;
                        when S3 =>
                                if hab_fila='1' then
                                        estado <= S0;
                                else
                                        estado <= S3;
                                end if;
                end case;
        end if;
end Process Maquina;

with estado select
        F0t <=  zeros when SA,
                fila3 when S0,
                fila0 when S1,
                fila1 when S2,
                fila2 when S3,
                zeros when others;

with estado select
        F1t <=  fila0 when SA,
                fila0 when S0,
                fila1 when S1,
                fila2 when S2,
                fila3 when S3,
                zeros when others;

with estado select
        F2t <=  fila1 when SA,
                fila1 when S0,
                fila2 when S1,
                fila3 when S2,
                fila0 when S3,
                zeros when others;
end arqui;
```

A.3. BLOQUE DE FILTRADO MORFOLÓGICO UNIDIMENSIONAL

FILTRADO_UNIDIM.VHD

```
library ieee;
use ieee.std_logic_1164.all;

USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

package filtrado_unidim_pack is

        component ordena_op
           generic (   ancho_palabra  : natural);
           port(       A_in : in      std_logic_vector(ancho_palabra - 1 downto 0);
                SE_in : in std_logic_vector(ancho_palabra*3 - 1 downto 0);
                clock, hab_colu, ini_fila, fin_fila  : in std_logic;
                A1_p, A2_p, A3_p : out std_logic_vector(ancho_palabra - 1 downto 0);
                SE1_p, SE2_p, SE3_p : out std_logic_vector(ancho_palabra - 1 downto 0));
        end component;

--Sumador Acotado
        component dila_sum
                generic ( ancho_palabra       : natural := 8);
                port(   A       : in    std_logic_vector(ancho_palabra - 1 downto 0);
                        B       : in    std_logic_vector(ancho_palabra - 1 downto 0);
                        clock   : in    std_logic;
                        SUMA    : out   std_logic_vector(ancho_palabra - 1 downto 0));
        end component;

--Restador Acotado
        component ero_sum
                generic ( ancho_palabra       : natural := 8);
                port(   A       : in    std_logic_vector(ancho_palabra - 1 downto 0);
                        B       : in    std_logic_vector(ancho_palabra - 1 downto 0);
                        clock   : in    std_logic;
                        RESTA   : out   std_logic_vector(ancho_palabra - 1 downto 0));
        end component;
        component mayor_pipe
                generic( ancho_palabra : natural := 8);
                port (
                        a, b, c : in    std_logic_vector (ancho_palabra-1 downto 0);
                        clock   : in    std_logic;
                        R       : out   std_logic_vector (ancho_palabra-1 downto 0)
                );
        end component;
        component menor_pipe
                generic( ancho_palabra : natural := 8);
                port (
                        a, b, c : in    std_logic_vector (ancho_palabra-1 downto 0);
                        clock   : in    std_logic;
                        R       : out   std_logic_vector (ancho_palabra-1 downto 0)
                );
        end component;
end filtrado_unidim_pack;

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.filtrado_unidim_pack.all;

entity filtrado_unidim is
        generic (       ancho_palabra  : natural := 8);
```

```
        port(   F_in : in      std_logic_vector(ancho_palabra - 1 downto 0);
                SE_in : in     std_logic_vector(ancho_palabra*3 - 1 downto 0);

                clock, hab_colu : in std_logic;

                ini_fila, fin_fila : in std_logic;

                DILATE, ERODE  : out   std_logic_vector(ancho_palabra - 1 downto 0));
end filtrado_unidim;

architecture arqui of filtrado_unidim is
        signal unos, zeros : std_logic_vector(ancho_palabra - 1 downto 0);

        signal SFa : std_logic_vector(ancho_palabra - 1 downto 0);
        signal A1p, A2p, A3p : std_logic_vector(ancho_palabra - 1 downto 0);
        signal A1t, A2t, A3t : std_logic_vector(ancho_palabra - 1 downto 0);

        signal SE1p, SE2p, SE3p : std_logic_vector(ancho_palabra - 1 downto 0);
        signal SE1t, SE2t, SE3t : std_logic_vector(ancho_palabra - 1 downto 0);

        signal SUMA1, SUMA2, SUMA3, RESTA1, RESTA2, RESTA3: std_logic_vector(ancho_palabra -
1 downto 0);

        signal ad, bd, cd, ae, be, ce : std_logic_vector(ancho_palabra - 1 downto 0);

begin

gen1:   for i in 0 to ancho_palabra-1 generate
                zeros(i) <= '0';
                unos(i) <= '1';
        end generate;

REG: Process(clock)
begin
        if (clock'EVENT and clock='1') THEN
                A1t <= A1p;     --REG2a
                A2t <= A2p; --REG2b
                A3t <= A3p; --REG2b

                SE1t <= SE1p;  --Registro SE1
                SE2t <= SE2p;  --Registro SE2
                SE3t <= SE3p;  --Registro SE3

                ad <= SUMA1;
                bd <= SUMA2;
                cd <= SUMA3;
                ae <= RESTA1;
                be <= RESTA2;
                ce <= RESTA3;

        end if;
end process REG;

order:  ordena_op
        generic map(ancho_palabra)
        port map(       A_in => F_in,
                        SE_in => SE_in,
                        clock => clock,
                        hab_colu => hab_colu,
                        ini_fila => ini_fila,
                        fin_fila => fin_fila,
                        A1_p => A1p, A2_p => A2p, A3_p => A3p,
                        SE1_p => SE1p, SE2_p => SE2p, SE3_p => SE3p);

-- *** BLOQUES QUE REALIZAN DILASION Y EROSION POR SEPARADO *** ---
dilasion_1 :    dila_sum
                generic map(ancho_palabra)
                port map(       A       => A1t,
                                B       => SE1t,
                                clock   => clock,
                                SUMA    => SUMA1);
```

```
erosion_1 :    ero_sum
               generic map(ancho_palabra)
               port map(      A       => A1t,
                              B       => SE1t,
                              clock   => clock,
                              RESTA   => RESTA1);

dilasion_2 :   dila_sum
               generic map(ancho_palabra)
               port map(      A       => A2t,
                              B       => SE2t,
                              clock   => clock,
                              SUMA    => SUMA2);
erosion_2 :    ero_sum
               generic map(ancho_palabra)
               port map(      A       => A2t,
                              B       => SE2t,
                              clock   => clock,
                              RESTA   => RESTA2);
dilasion_3 :   dila_sum
               generic map(ancho_palabra)
               port map(      A       => A3t,
                              B       => SE3t,
                              clock   => clock,
                              SUMA    => SUMA3);
erosion_3 :    ero_sum
               generic map(ancho_palabra)
               port map(      A       => A3t,
                              B       => SE3t,
                              clock   => clock,
                              RESTA   => RESTA3);
-- ****************************

-- Bloques donde se obtienen:
       -- máxima suma -> Dilación
dila_mayor:    mayor_pipe
               generic map (ancho_palabra)
               port map(ad, bd, cd, clock, DILATE);

       -- mínima resta -> Erosión
eros_menor:    menor_pipe
               generic map (ancho_palabra)
               port map(ae, be, ce, clock, ERODE);

end arqui;
```

A.4. BLOQUES ARITMÉTICOS

A.4.1. SUMADOR MORFOLÓGICO

DILA_SUM.VHD

```
library ieee;
use ieee.std_logic_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

entity dila_sum is
       generic ( ancho_palabra      : natural := 8);
       port(   A       : in    std_logic_vector(ancho_palabra - 1 downto 0);
               B       : in    std_logic_vector(ancho_palabra - 1 downto 0);

               clock   : in    std_logic;

               SUMA    : out   std_logic_vector(ancho_palabra - 1 downto 0)
               );
end dila_sum;

architecture arqui of dila_sum is
       signal over, A0, B0, A0a, B0a : std_logic;
       signal suma2, suma3, R_t : std_logic_vector(ancho_palabra - 1 downto 0);
       signal suma1 : std_logic_vector(ancho_palabra downto 0);
       signal unos, zeros : std_logic_vector(ancho_palabra - 1 downto 0);
begin
gen1:   for i in 0 to ancho_palabra-1 generate
               zeros(i) <= '0';
               unos(i) <= '1';
       end generate;

--Indicadores de nulidad
A0 <=   '1' when A=zeros else --Indica que el pixel es 0
       '0';
B0 <=   '1' when B=zeros else --Indica que el SE es 0
       '0';

suma1 <= A + B;

suma3 <=       suma2 when over='0' and A0a='0' and B0a='0' else
               unos when over='1' else
               zeros;

Registros: Process (clock)
begin
       if (clock'event and clock='1') then
               over <= suma1(ancho_palabra);
               suma2 <= suma1(ancho_palabra-1 downto 0);
               A0a <= A0;
               B0a <= B0;
               R_t <= suma3;
       end if;
end Process Registros;

SUMA <= r_t;

end arqui;
```

A.4.2. RESTADOR MORFOLÓGICO

ERO_SUM.VHD

```
library ieee;
use ieee.std_logic_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

entity ero_sum is
       generic ( ancho_palabra      : natural := 8);
       port(   A       : in    std_logic_vector(ancho_palabra - 1 downto 0);
               B       : in    std_logic_vector(ancho_palabra - 1 downto 0);
               clock   : in    std_logic;
               RESTA   : out   std_logic_vector(ancho_palabra - 1 downto 0) );
end ero_sum;

architecture arqui of ero_sum is
       signal over, A0, B0, A0a, B0a : std_logic;
       signal suma2, suma3, R_t : std_logic_vector(ancho_palabra - 1 downto 0);
       signal suma1 : std_logic_vector(ancho_palabra downto 0);
       signal unos, zeros : std_logic_vector(ancho_palabra - 1 downto 0);
Begin
gen1:  for i in 0 to ancho_palabra-1 generate
               zeros(i) <= '0';
               unos(i) <= '1';
       end generate;
--Indicadores de nulidad
A0 <=  '1' when A=zeros else --Indica que el pixel es 0
       '0';
B0 <=  '1' when B=zeros else --Indica que el SE es 0
       '0';
suma1 <= A - B;

suma3 <=       suma2 when over='0' and A0a='0' and B0a='0' else
               unos when B0a='1' else
               zeros;
Registros: Process (clock)
begin
       if (clock'event and clock='1') then
               over <= suma1(ancho_palabra);
               suma2 <= suma1(ancho_palabra-1 downto 0);
               A0a <= A0;
               B0a <= B0;
               R_t <= suma3;
       end if;
end Process Registros;
       RESTA <= r_t;
end arqui;
```

A.4.3. BLOQUE MÁXIMA SUMA

MAYOR_PIPE_VAR.VHD

```
library ieee;
use ieee.std_logic_1164.all;

entity mayor_pipe_var is
       generic( ancho_palabra : natural := 8);
       port (
               a, b    : in    std_logic_vector (ancho_palabra-1 downto 0);
               c_in    : in std_logic;
               salida  : out   std_logic);
end mayor_pipe_var;
```

```
architecture arqui of mayor_pipe_var is
       signal c : std_logic_vector (ancho_palabra-1 downto 0);
begin
  gen1: for i in 1 to ancho_palabra -1 generate
       c(i) <= (not c(i-1) and (a(i) and not b(i))) or (c(i-1) and (a(i) or not b(i)));
  end generate;
       c(0) <= (not c_in and (a(0) and not b(0))) or (c_in and (a(0) or not b(0)));

  salida <= c(ancho_palabra-1);

end arqui;
```

MAYOR_PIPE.VHD

```
library ieee;
use ieee.std_logic_1164.all;

package mayor_pipe_pack is
       component mayor_pipe_var
              generic( ancho_palabra : natural);
              port (  a, b    : in    std_logic_vector (ancho_palabra-1 downto 0);
                      c_in    : in std_logic;
                      salida  : out   std_logic);
       end component;
end mayor_pipe_pack;

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.mayor_pipe_pack.all;

entity mayor_pipe is
       generic( ancho_palabra : natural := 8);
       port (  a, b, c : in    std_logic_vector(ancho_palabra-1 downto 0);
              clock   : in    std_logic;
              R       : out   std_logic_vector(ancho_palabra-1 downto 0)   );
end mayor_pipe;

architecture arqui of mayor_pipe is
       signal A1, B1, A2, B2, C1, C2 : std_logic_vector(ancho_palabra-1 downto 0);
       signal CP1, CP2, RP, RP1, RP2, Rt : std_logic_vector(ancho_palabra-1 downto 0);
       signal mLSB1, mLSB2, mMSB1, mMSB2, mLSB3, mLSB4, mMSB3, mMSB4, cero : std_logic;
Begin
Registra: Process(clock)
begin
  if clock'event and clock='1' then
       A1 <= A;       A2 <= A1;
       B2 <= B1;      B1 <= B;
       C2 <= C1;      C1 <= C;
       CP2 <= CP1;            CP1 <= C2;
       RP1 <= RP;             RP2 <= RP1;
       mLSB2 <= mLSB1;                mMSB2 <= mMSB1;
       mLSB4 <= mLSB3;                mMSB4 <= mMSB3;
       R <= Rt;
  end if;
end Process Registra;

cero <= '0';

LSB_AB_RP: mayor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => A(3 downto 0),--ancho_palabra/2-1 downto 0),
                      b => B(3 downto 0),--ancho_palabra/2-1 downto 0),
                      c_in => cero,
                      salida => mLSB1 );
MSB_AB_RP: mayor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => A1(ancho_palabra-1 downto 4),--ancho_palabra/2),
```

```
                        b => B1(ancho_palabra-1 downto 4),--ancho_palabra/2),
                        c_in => mLSB2,
                        salida => mMSB1 );
with mMSB2 select
        RP <=   A2 when '1',
                B2 when others;
LSB_RPC_R: mayor_pipe_var
        generic map(ancho_palabra => 4)--ancho_palabra/2)
        port map(       a => RP(3 downto 0),
                        b => C2(3 downto 0),
                        c_in => cero,
                        salida => mLSB3 );
MSB_RPC_R: mayor_pipe_var
        generic map(ancho_palabra => 4)--ancho_palabra/2)
        port map(       a => RP1(ancho_palabra-1 downto 4),
                        b => CP1(ancho_palabra-1 downto 4),
                        c_in => mLSB4,
                        salida => mMSB3 );
with mMSB4 select
        Rt <=   RP2 when '1',
                CP2 when others;
end arqui;
```

A.4.4. BLOQUE MÍNIMA RESTA

MENOR_PIPE_VAR.VHD

```
library ieee;
use ieee.std_logic_1164.all;

entity menor_pipe_var is
        generic( ancho_palabra : natural := 4);
        port (
                a, b    : in    std_logic_vector (ancho_palabra-1 downto 0);
                c_in    : in std_logic;
                salida  : out   std_logic);
end menor_pipe_var;

architecture arqui of menor_pipe_var is
        signal c : std_logic_vector (ancho_palabra-1 downto 0);
begin
  gen1: for i in 1 to ancho_palabra -1 generate
        c(i) <= (not c(i-1) and (not a(i) and b(i))) or (c(i-1) and (not a(i) or b(i)));
  end generate;
        c(0) <= (not c_in and (not a(0) and b(0))) or (c_in and (not a(0) or b(0)));

  salida <= c(ancho_palabra-1);

end arqui;
```

MENOR_PIPE.VHD

```
library ieee;
use ieee.std_logic_1164.all;
package menor_pipe_pack is
        component menor_pipe_var
                generic( ancho_palabra : natural);
                port (  a, b    : in    std_logic_vector (ancho_palabra-1 downto 0);
                                c_in    : in std_logic;
                                salida  : out   std_logic);
        end component;
end menor_pipe_pack;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.menor_pipe_pack.all;
```

```
entity menor_pipe is
       generic( ancho_palabra : natural := 8);
       port (  a, b, c : in    std_logic_vector(ancho_palabra-1 downto 0);
               clock   : in    std_logic;
               R       : out   std_logic_vector(ancho_palabra-1 downto 0)   );
end menor_pipe;

architecture arqui of menor_pipe is
       signal A1, B1, A2, B2, C1, C2 : std_logic_vector(ancho_palabra-1 downto 0);
       signal CP1, CP2, RP, RP1, RP2, Rt : std_logic_vector(ancho_palabra-1 downto 0);
       signal mLSB1, mLSB2, mMSB1, mMSB2, mLSB3, mLSB4, mMSB3, mMSB4, cero : std_logic;
Begin
Registra: Process(clock)
begin
  if clock'event and clock='1' then
       A1 <= A;       A2 <= A1;
       B2 <= B1;      B1 <= B;
       C2 <= C1;      C1 <= C;
       CP2 <= CP1;            CP1 <= C2;
       RP1 <= RP;             RP2 <= RP1;
       mLSB2 <= mLSB1;               mMSB2 <= mMSB1;
       mLSB4 <= mLSB3;               mMSB4 <= mMSB3;
       R <= Rt;
  end if;
end Process Registra;

       cero <= '0';
LSB_AB_RP: menor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => A(3 downto 0),
                      b => B(3 downto 0),
                      c_in => cero,
                      salida => mLSB1 );
MSB_AB_RP: menor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => A1(ancho_palabra-1 downto 4),
                      b => B1(ancho_palabra-1 downto 4),
                      c_in => mLSB2,
                      salida => mMSB1 );
with mMSB2 select
       RP <=   A2 when '1',
                      B2 when others;
LSB_RPC_R: menor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => RP(3 downto 0),
                      b => C2(3 downto 0),
                      c_in => cero,
                      salida => mLSB3 );
MSB_RPC_R: menor_pipe_var
       generic map(ancho_palabra => 4)--ancho_palabra/2)
       port map(      a => RP1(ancho_palabra-1 downto 4),
                      b => CP1(ancho_palabra-1 downto 4),
                      c_in => mLSB4,
                      salida => mMSB3 );
with mMSB4 select
       Rt <=   RP2 when '1',
               CP2 when others;
end arqui;
```

A.5. ARQUITECTURA COMPLETA

MORFOVAR.VHD

```
library ieee;
use ieee.std_logic_1164.all;

USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

package morfovar_pack is

        component control_datos
        generic (       ancho_palabra  : natural;     --Niveles de grises
                        tamano : natural;                   --Ancho de la imagen máximo
                        total_filas : natural;        --Alto de la imagen máximo
                        n_filas : natural);                 --Numero de memorias internas
        port(   Dato_in : in std_logic_vector(ancho_palabra - 1 downto 0);
-- Señales de control de tamaño de la imagen
                width : in std_logic_vector(tamano-1 downto 0);
                heigth :  in std_logic_vector(total_filas-1 downto 0);
-- Señales de control del Chip
                clock, rst      : in std_logic;
-- Señales de control de Comunicacion--
                envio_dato      : in std_logic;
                prep_para_recib : out std_logic;
                dato_recibido : out std_logic;

                R_listo : out std_logic;
                R_read : in std_logic;
        -- Datos
                SF0_out, SF1_out, SF2_out, SF3_out : out std_logic_vector(ancho_palabra - 1
downto 0);
                SE1_out, SE2_out, SE3_out : out std_logic_vector(ancho_palabra*3 - 1 downto
0);
        -- Señales de control
                hab_col, hab_fila, last_row : out std_logic;
                inicio_fila, final_fila : out std_logic;
        -- Relacion de señales para VERIFICACIÓN
                j_out : out std_logic_vector(tamano-1 downto 0);
                i_out : out std_logic_vector(total_filas-1 downto 0);
                opj_out : out std_logic_vector(tamano-1 downto 0);
                opi_out : out std_logic_vector(total_filas-1 downto 0);
                estado_out : out std_logic_vector(2 downto 0)
                );
        end component;

        component ordena_filas
                generic (       ancho_palabra  : natural;
                                n_filas : natural);
                        --Cuatro memorias, una por cada fila temporal
                port(   M0, M1, M2, M3 : in   std_logic_vector(ancho_palabra - 1 downto 0);
                        clock, rst, hab_fila, low_row : in std_logic;
                        F0, F1, F2 : out std_logic_vector(ancho_palabra - 1 downto 0));
        end component;

        component filtrado_unidim
                generic (       ancho_palabra : natural);
                port(   F_in : in      std_logic_vector(ancho_palabra - 1 downto 0);
                        SE_in : in     std_logic_vector(ancho_palabra*3 - 1 downto 0);
                        clock, hab_colu : in std_logic;
                        ini_fila, fin_fila : in std_logic;
                        DILATE, ERODE  : out   std_logic_vector(ancho_palabra - 1 downto 0)
                        );
        end component;
```

```
	component mayor_pipe
		generic( ancho_palabra : natural);
		port (
			a, b, c : in	std_logic_vector (ancho_palabra-1 downto 0);
			clock	: in	std_logic;
			R	: out	std_logic_vector (ancho_palabra-1 downto 0)
		);
	end component;
	component menor_pipe
		generic( ancho_palabra : natural);
		port (
			a, b, c : in	std_logic_vector (ancho_palabra-1 downto 0);
			clock	: in	std_logic;
			R	: out	std_logic_vector (ancho_palabra-1 downto 0)
		);
	end component;

end morfovar_pack;

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.morfovar_pack.all;

entity morfovar is
	generic (	ancho_palabra  : natural := 8;		--niveles de grises
			tamano : natural := 8;			--ancho de la imagen
			total_filas : natural := 8;		--alto de la imagen
			n_filas : natural := 2);		--memorias internas
	port(
		-- BUSES DE ENTRADA
			PIXEL : in std_logic_vector(ancho_palabra-1 downto 0);
			BANDERA_IN	: in std_logic_vector(2 downto 0);
			SIZE : in std_logic_vector(total_filas+tamano-1 downto 0);

			CLOCK : in std_logic;

		-- BUSES DE SALIDA
			XY_MEM: out std_logic_vector(total_filas+tamano-1 downto 0);
			coordenadas : out std_logic_vector(total_filas+tamano-1 downto 0);
			BANDERA_OUT	: out std_logic_vector(5 downto 0);
			Dato_out	: out std_logic_vector(ancho_palabra*2- 1 downto 0)
		--La salida arroja tanto la dilasion como la erosion
		);
end morfovar;

architecture a of morfovar is

	signal fila0, fila1, fila2, fila3 : std_logic_vector(ancho_palabra - 1 downto 0);
	signal opera1, opera2, opera3 : std_logic_vector(ancho_palabra - 1 downto 0);
	signal DILA_fila1,DILA_fila2,DILA_fila3: std_logic_vector(ancho_palabra-1 downto 0);
	signal ERO_fila1,ERO_fila2,ERO_fila3: std_logic_vector(ancho_palabra - 1 downto 0);
	signal DILASION, EROSION : std_logic_vector(ancho_palabra - 1 downto 0);

	signal SE_1, SE_2, SE_3 : std_logic_vector(ancho_palabra*3 - 1 downto 0);

	signal zeros, unos : std_logic_vector(ancho_palabra - 1 downto 0);

	signal hab_X, hab_Y, ini_fil, fin_fil : std_logic;

	signal u_fila : std_logic; --señales de ultima fila
	signal c_fila : std_logic; --señales de CAMBIO DE FILA

begin

gen1:	for i in 0 to ancho_palabra-1 generate
		zeros(i) <= '0';
		unos(i) <= '1';
	end generate;
```

```
entrada_datos: control_datos
               generic map (ancho_palabra, tamano, total_filas, n_filas)
               port map (     Dato_in => PIXEL,
                              width => SIZE(tamano-1 downto 0),
                              heigth => SIZE(total_filas+tamano-1 downto tamano),
                      -- Señales de control del Chip
                              clock => CLOCK,
                              rst     => BANDERA_IN(2),
                      -- Señales de control de Comunicacion con la PC --
                              envio_dato => BANDERA_IN(0),
                              prep_para_recib => BANDERA_OUT(1),
                              dato_recibido => BANDERA_OUT(0),
                              R_listo => BANDERA_OUT(2),
                              R_read => BANDERA_IN(1),
                      -- Datos
                              SF0_out => fila0,
                              SF1_out => fila1,
                              SF2_out => fila2,
                              SF3_out => fila3,

                              SE1_out => SE_1,
                              SE2_out => SE_2,
                              SE3_out => SE_3,
                      -- Señales de control
                              hab_col => hab_X,
                              hab_fila => hab_Y,
                              last_row => u_fila,
                              inicio_fila => ini_fil,
                              final_fila => fin_fil,

                              j_out => XY_MEM(tamano-1 downto 0),
                              i_out => XY_MEM(total_filas+tamano-1 downto tamano),
                              opj_out => coordenadas(tamano-1 downto 0),
                              opi_out => coordenadas(total_filas+tamano-1 downto tamano),
                              estado_out => BANDERA_OUT(5 downto 3)
                      );

escoge_filas: ordena_filas
       generic map(ancho_palabra, n_filas)
       port map(      M0 => fila0,
                      M1 => fila1,
                      M2 => fila2,
                      M3 => fila3,
                      clock => CLOCK,
                      rst => BANDERA_IN(2),
                      hab_fila => hab_Y, low_row => u_fila,
                      F0 => opera1, F1 => opera2, F2 => opera3);

opera_fila1: filtrado_unidim
       generic map (ancho_palabra)
       port map (     F_in => opera1,
                      SE_in => SE_3,
                      clock => CLOCK,
                      hab_colu => hab_X,
                      ini_fila => ini_fil,
                      fin_fila => fin_fil,
                      DILATE => DILA_fila1,
                      ERODE => ERO_fila1);

opera_fila2: filtrado_unidim
       generic map (ancho_palabra)
       port map (     F_in => opera2,
                      SE_in => SE_2,
                      clock => CLOCK,
                      hab_colu => hab_X,
                      ini_fila => ini_fil,
                      fin_fila => fin_fil,
                      DILATE => DILA_fila2,
                      ERODE => ERO_fila2);

opera_fila3: filtrado_unidim
```

```
        generic map (ancho_palabra)
        port map (      F_in => opera3,
                        SE_in => SE_1,
                        clock => CLOCK,
                        hab_colu => hab_X,
                        ini_fila => ini_fil,
                        fin_fila => fin_fil,
                        DILATE => DILA_fila3,
                        ERODE => ERO_fila3);

-- BLOQUES DONDE SE OBTIENEN LOS MAXIMOS Y MINIMOS DE LOS RESULTADOS
-- PARCIALES DE EROSION Y DILACION DE CADA FILA
dila_mayor:    mayor_pipe
        generic map (ancho_palabra)
        port map(DILA_fila1, DILA_fila2, DILA_fila3, clock, DILASION);
eros_menor:    menor_pipe
        generic map (ancho_palabra)
        port map(ERO_fila1, ERO_fila2, ERO_fila3, clock, EROSION);

-- Se le asignan los resultados obtenidos en paralelo al bus de salida
Dato_out(ancho_palabra*2 - 1 downto ancho_palabra)  <=      DILASION;
Dato_out(ancho_palabra-1 downto 0)   <= EROSION;

end a;
```

A.6. ARQUITECTURA CON INTERFAZ E/S

A.6.1. CONTROL DE BANDERAS

CONTROL_BANDERAS.VHD

```
library ieee;
use ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity control_banderas is
        port(   B_IN : in std_logic_vector(3 downto 0);      --Señales del Bus
                adress : in    std_logic_vector(3 downto 0); --Direcciones

                b_salida : in std_logic_vector(2 downto 0);
                --Bandera de salida de la arquitectura

                envia, lee_bus, clock : in std_logic;

                b_entrada : out std_logic_vector(2 downto 0)
                );
end control_banderas;

architecture arqui of control_banderas is
-- Defino tipos para los estados
        type estado_tipo is (S0, S1, S2, S3);
        signal estado : estado_tipo;

        signal rst : std_logic;
        signal E_dato_t, Recibido_t, Lee_dila_t, Lee_ero_t: std_logic;
        signal E_dato, Recibido, Lee_dila, Lee_ero: std_logic;

--Define envio_dato y R_read que son necesario para la Arquitectura
        signal actuadores : std_logic_vector(1 downto 0);

        signal tiempo, tiempo_t, hab_temp, hab_temp_t, cl_temp, cl_temp_t : std_logic;
        signal temporiza: std_logic_vector(2 downto 0);
begin

rst <= B_in(2);

-- Señales de control sin Registrar
        E_dato_t <= '1' when envia='1' and b_salida(1)='1' else '0';
        Recibido_t <= '1' when b_salida(0)='1' and b_salida(2)='0' else '0';
        Lee_dila_t <= '1' when adress="0000" and b_salida(2)='1' and lee_bus='1' else '0';
        Lee_ero_t <= '1' when adress="0001" and b_salida(2)='1' and lee_bus='1' else '0';

        temporizador: lpm_counter
           GENERIC MAP (LPM_WIDTH => 3)
                PORT MAP (      aclr => cl_temp,
                                clock => clock,
                                cnt_en => hab_temp,
                                q => temporiza);

                tiempo_t <=    '1' when temporiza="110" or temporiza="111" else '0';
                hab_temp_t <='1' when estado=S3 and tiempo_t='0' else '0';
                cl_temp_t <= '1' when estado=S0 else '0';

-- Registro de señales de control
Registra: Process(clock)
begin
  if clock'event and clock='1' then
        E_dato <= E_dato_t;
        Recibido <= Recibido_t;
        Lee_dila <= Lee_dila_t;
        Lee_ero <= Lee_ero_t;
```

```
        tiempo <= tiempo_t;
        hab_temp <= hab_temp_t;
        cl_temp <= cl_temp_t;

  end if;
end Process Registra;

Maquina: Process(clock, rst)
begin
        if rst='1' then
                estado <= S0;
        elsif clock'event and clock='1' then
                case estado is
                        when S0 =>
                                if E_dato='1' then
                                        estado <= S1;
                                elsif Lee_dila='1' then
                                        estado <= S2;
                                else
                                        estado <= S0;
                                end if;
                        when S1 =>
                                if Recibido='1' then
                                        estado <= S0;
                                elsif Lee_dila='1' then
                                        estado <= S2;
                                else
                                        estado <= S1;
                                end if;
                        when S2 =>
                                if Lee_ero='1' then
                                        estado <= S3;
                                else
                                        estado <= S2;
                                end if;
                        when S3 =>
                                if tiempo = '1' then
                                        estado <= S0;
                                else
                                        estado <= S3;
                                end if;
                end case;
        end if;
end Process Maquina;

with estado select
        actuadores <=  "00" when S0,
                       "01" when S1,
                       "00" when S2,
                       "11" when S3,
                       "00" when others;

with B_in(3) select
        b_entrada(1 downto 0) <=     actuadores when '1',
                                     B_in(1 downto 0) when '0',
                                     "00" when others;

        b_entrada(2) <= rst;  --Asinación directa del RESET
end arqui;
```

A.6.2. INTERFAZ E/S

MM_GRIS.VHD

```
library ieee;
use ieee.std_logic_1164.all;

USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

package mm_gris_pack is

        component morfovar  --Declaracion del bloque de la Arquitectura
                generic (       ancho_palabra  : natural;
                                tamano : natural;
                                total_filas : natural;
                                n_filas : natural);
                port( -- BUSES DE ENTRADA
                        PIXEL : in std_logic_vector(ancho_palabra-1 downto 0);
                        SIZE : in std_logic_vector(total_filas+tamano-1 downto 0); --8x8
                        BANDERA_IN      : in std_logic_vector(2 downto 0);
                        CLOCK : in std_logic; -- infaltable
                -- BUSES DE SALIDA
                        XY_MEM: out std_logic_vector(total_filas+tamano-1 downto 0);
                        coordenadas : out std_logic_vector(total_filas+tamano-1 downto 0);

                        BANDERA_OUT     : out std_logic_vector(5 downto 0);
                        Dato_out        : out std_logic_vector(ancho_palabra*2- 1 downto 0)
                );
        end component;

        component control_banderas
        port(   B_IN : in std_logic_vector(3 downto 0);                 --Señales del Bus
                adress : in     std_logic_vector(3 downto 0); --Direcciones

                b_salida : in std_logic_vector(2 downto 0); --Bandera de salida de la
arquitectura

                envia, lee_bus, clock : in std_logic;

                b_entrada : out std_logic_vector(2 downto 0)
                );
        end component;

end mm_gris_pack;

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.mm_gris_pack.all;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

LIBRARY altera;
USE altera.maxplus2.ALL;

entity mm_gris is
        port(   sa : IN std_logic_vector(14 downto 0); --BUS DE DIRECCIONES
                sd      : INOUT std_logic_vector(15 downto 0); --BUS BIDIRECCIONAL DE DATOS

                clk : in std_logic; -- infaltable
        -- *** Señales de control ***
                --ENTRADAS
                memw, memr, bale : in std_logic;      --Control de escritura y lectura del bus
                flex_cs, nCS : in std_logic;          --Habilitador de FLEX

                bclk, flex_reset, reset, spare0, spare1 : in std_logic; --no usadas
```

```
                --SALIDAS
                nows, memcs16, int : out std_logic
                );
end mm_gris;

architecture a of mm_gris is
        signal zeros : std_logic_vector(15 downto 0);

-- Señales de control de la INTERFACE Bus Isa
        signal a : std_logic_vector(3 downto 0);
        signal reg8en_0, reg8en_1, reg8en_2, reg8en_3 : std_logic; --habilitadores de Latchs
        signal hab_lee : std_logic;

-- Señales que van al SISTEMA
        signal size, posicion, pos_mem, resultado : std_logic_vector(15 downto 0);
        signal DATO, senales : std_logic_vector(7 downto 0);
        signal controla : std_logic_vector(2 downto 0);
        signal control_previo : std_logic_vector(3 downto 0);

-- Señales que van al BUS BIDIRECCIONAL
        signal d_out, d_in : std_logic_vector(15 downto 0);

-- Multiplexores de salida
        signal  salida_10, salida_11, salida_12, salida_13 : std_logic_vector(7 downto 0);
        signal  salida_20, salida_21, salida8 : std_logic_vector(7 downto 0);
        signal  salida : std_logic_vector(15 downto 0);

        signal  R0, R1, R2, R3, R4, R5, R6, R7: std_logic_vector(7 downto 0);

begin

gen1:   for i in 0 to 15 generate
        zeros(i) <= '0';
end generate;

entrada_datos: morfovar
        generic map (   ancho_palabra => 8,     --niveles de grises
                        tamano => 8,            --ancho de la imagen
                        total_filas => 8,       --alto de la imagen
                        n_filas => 2)           --memorias internas
        port map (      PIXEL => DATO,
                        SIZE => size,
                        BANDERA_IN => controla,
                        CLOCK => clk,
                        XY_MEM => pos_mem,
                        coordenadas => posicion,
                        BANDERA_OUT => senales(5 downto 0),
                        Dato_out => resultado );

senales(7 downto 6) <= controla(1 downto 0);
--completo el bus con las señales de la Bandera de Entrada

--SEÑALES DE HABILITACION DE ESCRITURA EN LOS LATCH
reg8en_0 <= '1' when a="0000" and flex_cs='1' and memw='0' else '0'; --ojo memw ACTIVO
reg8en_1 <= '1' when a="0001" and flex_cs='1' and memw='0' else '0'; --en '0'
reg8en_2 <= '1' when a="0010" and flex_cs='1' and memw='0' else '0';
reg8en_3 <= '1' when a="0011" and flex_cs='1' and memw='0' else '0';

--SEÑAL DE HABILITACION DE LECTURA EN EL BUS
hab_lee <= '1' when memr='0' and flex_cs='1' else '0'; --memr ACTIVO en '0'

Bus_Datos : LPM_BUSTRI
   GENERIC MAP (LPM_WIDTH => 16)
   PORT MAP (  data => d_out,
               enabledt => hab_lee,
               result => d_in,
               tridata => sd);

with a select
salida8 <=      resultado(15 downto 8)when "0000",
                resultado(7 downto 0) when "0001",
```

```
                posicion(15 downto 8) when "0010",
                posicion(7 downto 0)  when "0011",
                senales               when "0100",
                DATO                  when "0101",
                pos_mem(15 downto 8)  when "0110",
                pos_mem(7 downto 0)   when "0111",
                size(15 downto 8)             when "1000",
                size(7 downto 0)              when "1001",
                zeros(7 downto 0)             when others;

d_out <= zeros(7 downto 0) & salida8;
 -- *** A LA SALIDA DEL BUS BIDIRECCIONAL ***

---------

Latch0: lpm_latch --LATCH DE CAPTURA DE LAS DIRECCIONES
   GENERIC MAP (LPM_WIDTH => 4)
   PORT MAP (  data => sa(3 downto 0),
               gate => bale,
               q => a);

-- **** SEÑALES DE ENTRADA AL SISTEMA A TRAVES DEL BUS ISA **** --
Latch1: lpm_latch --Bus de Datos de Entrada al Sistema
   GENERIC MAP (LPM_WIDTH => 8)
   PORT MAP (  data => d_in(7 downto 0),
               gate => reg8en_0,
               q => DATO);

Latch2: lpm_latch --Bus de señales de control al Sistema
   GENERIC MAP (LPM_WIDTH => 4)
   PORT MAP (  data => d_in(3 downto 0),
               gate => reg8en_1,
               q => control_previo);

--- Bloque de control de Banderas de entrada ---
c_previo: control_banderas
       port map (     B_IN => control_previo,
                      adress => a,
                      b_salida => senales(2 downto 0),
                      envia => reg8en_0,
                      lee_bus => hab_lee,
                      clock => clk,
                      b_entrada => controla );
---------------

Latch3: lpm_latch -- ANCHO DE LA IMAGEN
   GENERIC MAP (LPM_WIDTH => 8)
   PORT MAP (  data => d_in(7 downto 0),
               gate => reg8en_2,
               q => size(7 downto 0));

Latch4: lpm_latch -- ALTURA
   GENERIC MAP (LPM_WIDTH => 8)
   PORT MAP (  data => d_in(7 downto 0),
               gate => reg8en_3,
               q => size(15 downto 8));

-- Señales de Control de Salida
nows <= not flex_cs;
memcs16 <= '1';              --Esto indica que nunca se trabajara con Registros de 16 bits
int <= zeros(0);      --Interrupciones deshabilitadas

end a;
```

ANEXO B: CÓDIGO DEL PROGRAMA DE CONTROL

B.1. PROGRAMA PRINCIPAL

```
#include "DEFINES.h"

int main()
{
 int seleccion=0;
 char indica=0;
 char nombre[20];

 while (seleccion != ESC)
 {
  clrscr();
  printf("\n ********* FILTRO MORFOLOGICO EN UN FPGA DE ALTERA *********");
  printf("\n ****** Programa de control ******* \n\n");
  printf("\n   1. Abrir Imagen BMP\n");
  printf("\n   2. Ingresar Elemento Estructural\n");
  printf("\n   3. Prueba Funcional \n");
  printf("\n   4. Prueba de Rendimiento \n");
  printf("\n   5. Procesar Imagen \n");
  printf("\n   6. Convertir Resultados a BMP \n");
  printf("\n\n       ** Presione ESC para salir **\n");

  seleccion=getch();

  if (seleccion==49)       //opcion 1
   {
          clrscr();

          printf("Ingrese nombre de la imagen: ");
          scanf("%s",nombre);

          indica = bmp_hex(nombre); //Convierte BMP a hexadecimal

           if (indica==1)
            { printf("\nArchivo no pudo abrirse\n");
              getch();  }

           if (indica==2)

            { printf("Archivo no es BMP");
              getch();  }

           if (indica==3)
            { printf("Archivo no est  en Escala de Grises");
              getch();  }

           if (indica==4)
            { printf("El tamaño no es apropiado");
              getch();  }
   }
```

```
  if (seleccion==50) pide_SE(); /*Opcion 2*/

  if (seleccion==51) mm_gray(); /*Opcion 3*/

  if (seleccion==52) p_tiempo(); /*Opcion 4*/

  if (seleccion==53) mm_fast(); /*Opcion 5*/

  if (seleccion==54)       //opcion 6
   {
          clrscr();
          printf("Nombre para el archivo de Dilación (BMP) : ");
          scanf("%s",nombre);
          indica = hex_bmp("DILA.HEX",nombre);

           if (indica==1) printf("\n Archivo de cabezera no encontrado\n");

           if (indica==2) printf(" No se puede crear archivo BMP");

           if (indica==3) printf(" Error. Fuente de Datos no encontrado");

          printf("Nombre para el archivo de Erosión (BMP) : ");
          scanf("%s",nombre);
          indica = hex_bmp("ERO.HEX",nombre);

           if (indica==1)
            { printf("\n Archivo de cabezera no encontrado\n");
              getch();  }

           if (indica==2)

            { printf(" No se puede crear archivo BMP");
              getch();  }

           if (indica==3)
            { printf(" Error. Fuente de Datos no encontrado");
              getch();  }
    }
 }
 return 0;
}
```

B.2. LIBRERÍA DEFINES.H

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#include <sys\stat.h>
#include <fcntl.h>
#include <io.h>

#include <dos.h>
#include <time.h>  //Para manejar el cronometro
#include <string.h> //Para el archivo reporte

unsigned BaseAddress;

int width=0, heigth=0;

const int True=1;
const int False=0;
const unsigned ConfigOffset = 0x2000;
enum op {set, restore};
char shadowreg = 0x00; // este es usado para mantener 'keep track of the state
                       // of' los registros de configuracion. Aquellos
                       // registros no pueden ser leidos  como se describe
                       // en el Datasheet de la Constellation.

#define ESC   0x1b //Solo defino codigos de teclas como constantes
#define ENTER 0x0d //para utilizarlas luego


////////////////////////////////////
//Registros de entrada de 1 byte//
////////////////////////////////////
   unsigned PIXEL = 0x0000;
   unsigned BANDERA_IN = 0x0001;
   unsigned ANCHO = 0x0002;
   unsigned ALTURA = 0x0003;
////////////////////////////////////

////////////////////////////////////////////////
//Declaracion de Banderas y Registros de estado//
////////////////////////////////////////////////
unsigned DILA = 0x0000;
unsigned ERO = 0x0001;
unsigned OPI_OUT = 0x0002;
unsigned OPJ_OUT = 0x0003;

unsigned BANDERA_OUT=0x0004;
unsigned CHK_DATO = 0x0005;
unsigned I_OUT = 0x0006;
unsigned J_OUT = 0x0007;
unsigned CHK_WIDTH = 0X0008;
unsigned CHK_HEIGTH = 0X0009;
```

```
//////////////////////////////////////////////////

   char B_IN;  // Señales de entrada. Control y Comunicacion

   char data_in;

   int data, i_out, j_out, opi_out, opj_out;


// Variables utilizadas para la comparación //
   unsigned char mo_dila, mo_ero, ana_dila, ana_ero;
   unsigned int n_fallas_dila, n_fallas_ero;
//////////////////////////////////////////////////

#include "in_out.h"
#include "entra_da.h"
#include "banderas.h"
#include "mm_gray.h"
#include "p_tiempo.h"
#include "mm_fast.h"
#include "bmp_lib.h"
```

B.3. LIBRERÍA IN_OUT.H

```
// Subrutinas de escritura en el Bus ISA

void Read(unsigned BaseAddress,unsigned SegmentOffset, int &data);
void Read(unsigned BaseAddress, unsigned SegmentOffset, char &data);
void Write(unsigned BaseAddress, unsigned SegmentOffset, int data);
void Write(unsigned BaseAddress, unsigned SegmentOffset, char data);

/*********************************************/

void Read(unsigned BaseAddress,unsigned SegmentOffset, int &data)
{  data = peek(BaseAddress,SegmentOffset); }

void Read(unsigned BaseAddress, unsigned SegmentOffset, char &data)
{  data = peekb(BaseAddress,SegmentOffset); }

void Write(unsigned BaseAddress, unsigned SegmentOffset, int data)
{  poke(BaseAddress,SegmentOffset,data); }

void Write(unsigned BaseAddress, unsigned SegmentOffset, char data)
{  pokeb(BaseAddress,SegmentOffset,data); }
```

B.4. LIBRERÍA ENTRA_DA.H

```
int pide_SE(void);
int pide_A(void);

unsigned char num;
/*****************************************/

int pide_SE(void)
{
  FILE *fp1;

  int i=1,j=0;

  clrscr();

// Secuencia de ESCRITURA
  if ( (fp1 = fopen("se.ejm", "wb")) == NULL)
  {
     fprintf(stderr, "\nError al abrir archivo\n");
     return 1;
            }
  printf("\n Ingrese los numeros del SE a grabar\n");

  j=0; i=0;

  while(i<3)
  {
   gotoxy(1,3);
   printf("                                  ");
   gotoxy(1,3);
   printf("SE (%d,%d) : ",j,i);
   scanf("%d",&num);

   gotoxy(5+j*5,5+i*2);
   printf("%d",num);

   if (j==2) {j=0; i++;} else j++;

   fwrite(&num,1,1,fp1);
  }

  fclose(fp1);

// Secuencia de Lectura
  if ( (fp1 = fopen("se.ejm", "rb")) == NULL)
  {  fprintf(stderr, "\nError al abrir archivo\n");
     return 2; }

  clrscr();
  printf("\n *** ELEMENTO ESTRUCTURAL INGRESADO*** \n");

  i=0; j=0;
  while (i<3)
```

```
  {
     fread(&num,sizeof(char),1,fp1);
     gotoxy(5+j*5,5+i*2);
     printf("%d",num);

     if (j==2) {j=0; i++;} else j++;
  }
  fclose(fp1);

  getch();
  return 0;
}

int pide_A(void)
{
  FILE *fp1;

//  unsigned char num;
  int i=1,j=0;

  clrscr();
  width=0;
  while( (width < 5) | (width > 12))
   { printf("\n Ingrese Ancho : ");
     scanf("%d",&width);
   }
  heigth=0;
  while( (heigth < 5) | (heigth > 12) )
   { printf("\n Ingrese Altura : ");
     scanf("%d",&heigth);
   }

  clrscr();

// Secuencia de ESCRITURA
  if ( (fp1 = fopen("A.ejm", "wb")) == NULL)
  {
     fprintf(stderr, "\nError al abrir archivo\n");
     return 1;
  }

  width--;
  heigth--;

  fwrite(&width,1,1,fp1);
  fwrite(&heigth,1,1,fp1);

  gotoxy(1,1);
  printf("\n Ingrese los numeros de la Imagen a grabar\n");

  j=0; i=0;
  while(i < heigth+1)
  {
   gotoxy(1,3);
   printf("                                    ");
   gotoxy(1,3);
```

```
   printf("Pixel (%d,%d) : ",j,i);
   scanf("%d",&num);

   gotoxy(5+j*5,5+i*2);
   printf("%d",num);

   fwrite(&num,1,1,fp1);

   if (j==width) {j=0; i++;} else j++;
  }

  fclose(fp1);

  return 0;
}
```

B.5. LIBRERÍA BANDERAS.H

```
   char recibido=0;        // Dato_recibido
   char preparado=0;       // Preparado para recibir dato
   char R_listo=0;         // Resultado listo para ser capturado

   char estado_FLEX=0;// Indica el estado en que se encuentra el sistema
   char B_in_leido=0;      // Se puede ver la bandera ingresada al sistema

   int B_OUT=0;

void lee_banderas(void);
void muestra_banderas(void);

// ESTAs SUBRUTINAs SOLO SEPARA LAS BANDERAS Y LAS MUESTRA EN PANTALLA
void lee_banderas(void)
{
   Read(BaseAddress,BANDERA_OUT,
   B_OUT); // Lee el registro de estados

     B_OUT = B_OUT & 0x00FF;

     estado_FLEX = (B_OUT >> 3) & 0x07;

     recibido = B_OUT & 0x01;

     preparado = (B_OUT >> 1) & 0x01;

     R_listo = (B_OUT >> 2) & 0x01;

     B_in_leido = (B_OUT >> 6) & 0x03;

   Read(BaseAddress,CHK_WIDTH,width); //Comprobación del registro de ancho
   width=width & 0x00FF;

   Read(BaseAddress,CHK_HEIGTH,heigth);
   heigth = heigth & 0x00FF;   }
```

```
void muestra_banderas(void)
{  gotoxy(1,4);
   printf("BANDERAS DE ESTADO. FLEX INDICA: ");

   gotoxy(3,7);
   printf("Dato Recibido = %d   ",recibido);

   gotoxy(3,9);
   printf("Preparado para recibir Dato = %d   ",preparado);

   gotoxy(3,11);
   printf("Resultado listo = %d   ",R_listo);

   gotoxy(3,13);
   printf("Estado = %d   ",estado_FLEX);

   gotoxy(3,15);
   printf("Bandera de Entrada = %d   ",B_in_leido);

   gotoxy(3,16);
   printf("Tamaño de imagen: ancho=%d alto=%d",width+1,heigth+1);
}
```

B.6. LIBRERÍA MM_GRAY.H

```
int mm_gray();

int mm_gray()

{
   unsigned long tempaddr;
   char tecla;

//DECLARACION DE PUNTEROS DE ARCHIVO
   FILE *fp_SE, *fp_A, *fp_DILA, *fp_ERO;

   FILE *fp_mdil, *fp_mero;

   FILE *stream;

   clrscr();

// Direccion BASE
   tempaddr = 0xcc000 >> 4;
   BaseAddress = (unsigned) tempaddr;

// DECLARACION DE VARIABLES
   char B_IN;  // Señales de entrada. Control y Comunicacion
   char data_in;

   int data, i_out, j_out, opi_out, opj_out;

   char dila=0;
   char ero=0;    //Resultados del Sistema separados
```

```
  int j=0,i=0;
  int j_SE=0,i_SE=0;

  struct  time t1, t2; //Variables que miden la diferencia de tiempo

/*******************************
** INICIO DEL PROCESO **
*******************************/

 clrscr();
 char muestra=0, pausa=0;

  printf("\n MOSTRAR REGISTROS Paso a Paso? (SI=ENTER):");
  tecla = getch();
  if (tecla==ENTER) {muestra=1; pausa=1;} else {muestra=0; pausa=0;}
 ////////////////////////////////////////////////

  printf("\n Ingresar una nueva Matriz Pequeña ? (NO=ENTER):");
  tecla = getch();
  if (tecla!=ENTER) pide_A();

  printf("\n Desea Ingresar un nuevo SE ? (NO=ENTER):");
  tecla = getch();
  if (tecla!=ENTER) pide_SE();


// *** INICIO DEL BUCLE DE PRUEBAS ***
 fp_SE = fopen("se.ejm", "rb");

  fp_A = fopen("A.ejm", "rb");
  fp_DILA = fopen("DILA.ejm", "wb");
  fp_ERO = fopen("ERO.ejm", "wb");

/* IMPORTANTE: AQUI SE INGRESA EL TAMANO DE LA IMAGEN   AL SISTEMA */
 fread(&width,1,1,fp_A);
 fread(&heigth,1,1,fp_A);

 fwrite(&width,1,1,fp_DILA);
 fwrite(&heigth,1,1,fp_DILA);

 fwrite(&width,1,1,fp_ERO);
 fwrite(&heigth,1,1,fp_ERO);

 Write(BaseAddress,ANCHO,width); // ANCHO

 Write(BaseAddress,ALTURA,heigth); // ALTO

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

 B_IN = 0;//  Deshabilito el PIN de RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads


/* ES IMPORTANTE LEER LAS BANDERAS PARA SABER QUE EL PROCESO SE REINICIO*/
```

```
 j=0; i=0;
 j_SE=0; i_SE=0;

 data=0; j_out=0; i_out=0; opi_out=0; opj_out=0;

 clrscr();
 gotoxy(1,1);
 printf("\n *** Pruebas Funcionales ***");

 dila=0;
 ero=0;

 lee_banderas();

 gettime(&t1); //lee tiempo inicial

while(estado_FLEX !=7)
{
  if (muestra==1) muestra_banderas();

// Bloques de analisis
  if (i_SE<3)  //Secuencia de entrada del SE
   {
     if (preparado==1)
     {
            fread(&data_in,sizeof(char),1,fp_SE); //sizeof(char)=1 mide el tamaño de un char
            Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado*/

            B_IN =0x01;
            Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

            B_IN =0x00;
            Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estado

          ///////
           if (muestra==1)
           {Read(BaseAddress,CHK_DATO,data); // Leo el dato del registro ingresado
            gotoxy(15,19);
            printf("                  ");
            gotoxy(1,19);
            printf("SE Ingresado (%d,%d) : %d",j_SE,i_SE,data_in);
           }
            if (muestra==1) { lee_banderas(); muestra_banderas();}
            if (pausa==1) getch();
          ///////

            if (j_SE==2) {j_SE=0; i_SE++;} else j_SE++;
     }
   }
   else
    {  if (muestra==1)
            {
             Read(BaseAddress,CHK_DATO,data); // Leo el dato del registro ingresado
             Read(BaseAddress,J_OUT,j_out);
             Read(BaseAddress,I_OUT,i_out);
             data = data & 0x00FF;
```

```
            j_out = j_out & 0x00FF;
            i_out = i_out & 0x00FF;
            gotoxy(15,20);
            printf("                ");
            gotoxy(1,20);
            printf("Dato Ingresado (%d,%d) : %d",i_out,j_out,data);

            //Muestra los resultados
            Read(BaseAddress,OPJ_OUT,opj_out); // Leo el resultados
            Read(BaseAddress,OPI_OUT,opi_out);
            opj_out = opj_out & 0x00FF;
            opi_out = opi_out & 0x00FF;

            gotoxy(1,21);
            printf("Resultado. Fila:%d  Columna:%d ",opi_out,opj_out);

            gotoxy(1,22);
            printf("Dilación : %d      erosión: %d ",dila,ero);
           }
          if (pausa==1) tecla=getch();


        if ( i<heigth+1 )//& preparado==1 )
         {
          fread(&data_in,sizeof(char),1,fp_A);
          Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado

          if (j==width) {j=0; i++;} else j++;
          B_IN=0x01;

         }

        if ( R_listo==1)
         {
          Read(BaseAddress,DILA,dila); // Leo el resultados
          Read(BaseAddress,ERO,ero);
          dila= dila & 0x00FF;
          ero = ero & 0x00FF;

          fwrite(&dila,1,1,fp_DILA);
          fwrite(&ero,1,1,fp_ERO);

          B_IN =0x03; //Le dice que leyo el resultado
         }

        //////
          Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

          if (muestra==1) { lee_banderas(); muestra_banderas();}
          if (pausa==1) tecla=getch();

          B_IN =0x00;
          Write(BaseAddress,BANDERA_IN,B_IN);/// Escribe en el Registro de estado
 }
        ///////
lee_banderas();
```

```
}

gettime(&t2);

B_IN = 4; // Activo el PIN de RESET
Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

fclose(fp_SE);
fclose(fp_A);

fclose(fp_DILA);
fclose(fp_ERO);

  gotoxy(1,23);
  printf("Hora: Inicio=%2d:%02d:%02d      Finalizacion=%2d:%02d:%02d",
  t1.ti_hour, t1.ti_min, t1.ti_sec, t2.ti_hour, t2.ti_min, t2.ti_sec);

// *** BLOQUE DE OBTENCION DE RESULTADOS ***

  //MUESTRO DILASION
  fp_DILA = fopen("DILA.ejm", "rb");

  // Leo primero los dos bytes de tamaño
  fread(&width,1,1,fp_DILA);
  fread(&heigth,1,1,fp_DILA);

  clrscr();
  printf("\n *** DILASION *** \n");

  i=0; j=0;

  while (i < heigth+1)
   {
     fread(&num,sizeof(char),1,fp_DILA);

     gotoxy(5+j*5,5+i*2);
     printf("%d",num);

     if (j==width) {j=0; i++;} else j++;
   }

  fclose(fp_DILA);
  tecla=getch();

  //MUESTRO EROSION
  fp_ERO = fopen("ERO.ejm", "rb");

  // Leo primero los dos bytes de tamaño
  fread(&width,1,1,fp_ERO);
  fread(&heigth,1,1,fp_ERO);

  clrscr();
  printf("\n *** EROSION *** \n");

  i=0; j=0;
```

```
  while (i < heigth+1)
   {
     fread(&num,sizeof(char),1,fp_ERO);

     gotoxy(5+j*5,5+i*2);
     printf("%d",num);

     if (j==width) {j=0; i++;} else j++;
   }

   fclose(fp_ERO);

   tecla=getch();

//////////////////////////////////////////////

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

/*******************************
*** FIN DEL PROCESO ****
*******************************/

 printf("\n FIN DE LAS PRUEBAS");
 tecla = getch(); //solo para mantener la vista de la Pantalla
 return 0;
}
```

B.7. LIBRERÍA P_TIEMPO.H

```
int p_tiempo();

int p_tiempo()

{
  unsigned long tempaddr;

  char tecla;

//DECLARACION DE PUNTEROS DE ARCHIVO
  FILE *fp_SE, *fp_A, *fp_DILA, *fp_ERO;

  FILE *fp_mdil, *fp_mero;

  FILE *stream;

  clrscr();

  char dila=0;
  char ero=0;    //Resultados del Sistema separados

  int j=0,i=0;
  int j_SE=0,i_SE=0;
```

```
   int j_im=0, i_im=0, ya=0;

///////////////Declaracion de la direccion
   tempaddr = 0xcc000 >> 4;
   BaseAddress = (unsigned) tempaddr;
//////////////////////////////////////////

// DECLARACION DE VARIABLES

   struct  time t1, t2; //Variables que miden la diferencia de tiempo

   float demora, t_promedio;

   unsigned int num=0, max=0;

// Para el Archivo de reporte
   int dila_fail=0, ero_fail=0; //variables que indican falla en el sistema

   char ok[]= "OK  ";
   char falla[]= "FALLA";

/*
                    ******************************
                    ***** INICIO DEL PROCESO *****
                    ******************************
*/

clrscr();
char muestra=0, pausa=0;

   printf("\n CUANTAS PRUEBAS QUIERES REALIZAR? ");
   scanf("%d",&max);

   stream = fopen("reporte.txt", "wb");
   fprintf(stream,"**** REPORTE DE PRUEBAS DE LA ARQUITECUTURA  ****\n");
   fprintf(stream,"\n-n§-  DILA       ERO\n");

  mo_dila=0; mo_ero=0; ana_dila=0; ana_ero=0;
  n_fallas_dila=0; n_fallas_ero=0;


 printf("\n\n Presione ENTER para INICIAR PRUEBAS");
 getch();

  gettime(&t1); //lee tiempo inicial

// *** INICIO DEL BUCLE DE PRUEBAS ***
while (num<max)
{
 fp_SE = fopen("se.ejm", "rb");

   fp_A = fopen("A.hex", "rb");

   fp_DILA = fopen("DILA.hex", "wb");
   fp_ERO = fopen("ERO.hex", "wb");
```

```
/* IMPORTANTE: AQUI SE INGRESA EL TAMANO DE LA IMAGEN   AL SISTEMA */
 fread(&width,1,1,fp_A);
 fread(&heigth,1,1,fp_A);

 fwrite(&width,1,1,fp_DILA);
 fwrite(&heigth,1,1,fp_DILA);

 fwrite(&width,1,1,fp_ERO);
 fwrite(&heigth,1,1,fp_ERO);

 Write(BaseAddress,ANCHO,width); // ANCHO

 Write(BaseAddress,ALTURA,heigth); // ALTO

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estado


 ////////////////////
 B_IN = 8;                    //  habilito modo RAPIDO
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estado
 ////////////////////

 j=0; i=0;
 j_SE=0; i_SE=0;

 data=0; j_out=0; i_out=0; opi_out=0; opj_out=0;

 j_im=0; i_im=0;

 clrscr();
 gotoxy(1,1);
 printf("\n *** MORFOVAR ***  Prueba #: %d   Total de Pruebas: %d\n\n",num+1,max);

 dila=0;
 ero=0;

 dila_fail=0;
 ero_fail=0;

 ya=0;

 while (i_im < heigth+1)
 {
   if (muestra==1) muestra_banderas();

// Bloques de analisis
   if (i_SE<3)  //Secuencia de entrada del SE
    {
      fread(&data_in,sizeof(char),1,fp_SE); //sizeof(char)=1 mide el tamaño de un char
      Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado*/

      if (muestra==1)
          {Read(BaseAddress,CHK_DATO,data); // Leo el dato del registro ingresado
           gotoxy(15,19);
           printf("        ");
```

```
          gotoxy(1,19);
          printf("SE Ingresado (%d,%d) : %d",j_SE,i_SE,data_in);
         }
    if (muestra==1) { lee_banderas(); muestra_banderas();}
    if (pausa==1) getch();

           if (j_SE==2) {j_SE=0; i_SE++;} else j_SE++;
 }
else
 {  if (muestra==1)
           {
            Read(BaseAddress,CHK_DATO,data); // Leo el dato del registro ingresado
            Read(BaseAddress,J_OUT,j_out);
            Read(BaseAddress,I_OUT,i_out);
            data = data & 0x00FF;
            j_out = j_out & 0x00FF;
            i_out = i_out & 0x00FF;
            gotoxy(15,20);
            printf("                ");
            gotoxy(1,20);
            printf("Dato Ingresado (%d,%d) : %d",i_out,j_out,data);

            //Muestra los resultados
            Read(BaseAddress,OPJ_OUT,opj_out); // Leo el resultados
            Read(BaseAddress,OPI_OUT,opi_out);
            opj_out = opj_out & 0x00FF;
            opi_out = opi_out & 0x00FF;

            gotoxy(1,21);
            printf("Resultado. Fila:%d  Columna:%d ",opi_out,opj_out);

            gotoxy(1,22);
            printf("Dilasion : %d      erosion: %d ",dila,ero);
           }
           if (pausa==1) getch();


         if ( i<heigth+1 )//& preparado==1 )
         {
          fread(&data_in,sizeof(char),1,fp_A);
          Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado

          if (j==width) {j=0; i++;} else j++;
         }

         if (i==2 & j==2) ////////////////////////////////////////////////
          {ya=1;}

         if ( (ya == 1) & (i_im < heigth +1) )
         {
          Read(BaseAddress,DILA,dila); // Leo el resultados
          Read(BaseAddress,ERO,ero);

          dila= dila & 0x00FF;
          ero = ero & 0x00FF;
```

```
            fwrite(&dila,1,1,fp_DILA);
            fwrite(&ero,1,1,fp_ERO);

            if (j_im==width) {j_im=0; i_im++;} else j_im++;
           }

            if (muestra==1) { lee_banderas(); muestra_banderas();}
            if (pausa==1) getch();
   }
 }

 B_IN = 4; // Activo el PIN de RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

 fclose(fp_SE);
 fclose(fp_A);

 fclose(fp_DILA);
 fclose(fp_ERO);

// *** BLOQUE DE OBTENCION DE RESULTADOS ***

   // *** VERIFICACION DE ERRORES de la DILACION ***
   fp_mdil = fopen("M_DILA.hex", "rb");
   fp_DILA = fopen("DILA.hex", "rb");

   while( (!feof(fp_mdil)) & (dila_fail==0) )
   {
     fread(&mo_dila,1,1,fp_mdil);
     fread(&ana_dila,1,1,fp_DILA);

     if (mo_dila!=ana_dila) dila_fail=1; else dila_fail=0;
   }
   fclose(fp_mdil);
   fclose(fp_DILA);

   // *** VERIFICACION DE ERRORES de la EROSION ***
   fp_mero = fopen("M_ERO.hex", "rb");
   fp_ERO = fopen("ERO.hex", "rb");

   while( (!feof(fp_mero)) & (ero_fail==0) )
   {
     fread(&mo_ero,1,1,fp_mero);
     fread(&ana_ero,1,1,fp_ERO);

     if (mo_ero!=ana_ero) ero_fail=1; else dila_fail=0;
    }
   fclose(fp_mero);
   fclose(fp_ERO);

   // Escribo valores en el archivo de texto
   fprintf(stream, "\n%d      ", num+1); //numero de prueba

   if (dila_fail==0)
     fwrite(ok, strlen(ok)+1,1,stream);
   else
```

```
    { fwrite(falla, strlen(falla)+1,1,stream);
      n_fallas_dila++; }
      fprintf(stream,"      ");            // Tiempo de duracion

    if (ero_fail==0)
      fwrite(ok, strlen(ok)+1,1,stream);
    else
      { fwrite(falla, strlen(falla)+1,1,stream);
            n_fallas_ero++; }

////////////////////////////////////////////

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

 num++;

}// FIN DEL BUCLE DE PRUEBAS

 gettime(&t2);
 demora=1000*(3600*(t2.ti_hour-t1.ti_hour)+60*(t2.ti_min - t1.ti_min)+(t2.ti_sec -
t1.ti_sec))+10*(t2.ti_hund-t1.ti_hund);

 t_promedio=demora/max;

/******** FIN DEL PROCESO *******/

 fprintf(stream,"\n\n\n*** TIEMPO DE PROCESAMIENTO PROMEDIO= %.3f",t_promedio);

 fprintf(stream,"\n\n  HORA DE FINALIZACION :%2d:%02d",t2.ti_hour, t2.ti_min);

 fprintf(stream,"\n\n Numero de fallas en Dilasion = %d",n_fallas_dila);
 fprintf(stream,"\n Numero de fallas en Erosion = %d",n_fallas_ero);

 fprintf(stream,"\n\n ********** FIN DE REPORTE **********");
 fclose(stream); //cierro archivo reportes

 printf("\n FIN DE LAS PRUEBAS");
 getch();
 return 0;
}
```

B.8. LIBRERÍA MM_FAST.H

```
int mm_fast();

int mm_fast()
{
  unsigned long tempaddr;
  char tecla;

//DECLARACION DE PUNTEROS DE ARCHIVO
  FILE *fp_SE, *fp_A, *fp_DILA, *fp_ERO;

  FILE *fp_mdil, *fp_mero;

  FILE *stream;

  char dila=0;
  char ero=0;    //Resultados del Sistema separados

  int j=0,i=0;
  int j_SE=0,i_SE=0;

  int j_im=0, i_im=0, ya=0;

///////////////Declaracion de la direccion
  tempaddr = 0xcc000 >> 4;
  BaseAddress = (unsigned) tempaddr;
//////////////////////////////////////////

// DECLARACION DE VARIABLES

  struct  time t1, t2; //Variables que miden la diferencia de tiempo

  float demora, t_promedio;

  unsigned int num=0, max=0;

// Para el Archivo de reporte
  int dila_fail=0, ero_fail=0; //variables que indican falla en el sistema

  char ok[]= "OK  ";
  char falla[]= "FALLA";

/**********************************
*** INICIO DEL PROCESO ***
**********************************/

//char muestra=0, pausa=0;


 mo_dila=0; mo_ero=0; ana_dila=0; ana_ero=0;
 n_fallas_dila=0; n_fallas_ero=0;

 gettime(&t1); //lee tiempo inicial
```

```
// *** INICIO DEL BUCLE DE PRUEBAS ***

 fp_SE = fopen("se.ejm", "rb");

   fp_A = fopen("A.hex", "rb");

   fp_DILA = fopen("DILA.hex", "wb");
   fp_ERO = fopen("ERO.hex", "wb");

/* IMPORTANTE: AQUI SE INGRESA EL TAMANO DE LA IMAGEN   AL SISTEMA */
 fread(&width,1,1,fp_A);
 fread(&heigth,1,1,fp_A);

 fwrite(&width,1,1,fp_DILA);
 fwrite(&heigth,1,1,fp_DILA);

 fwrite(&width,1,1,fp_ERO);
 fwrite(&heigth,1,1,fp_ERO);

 Write(BaseAddress,ANCHO,width); // ANCHO
 Write(BaseAddress,ALTURA,heigth); // ALTO

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estado

 ////////////////////////////////////////////////////////////
 B_IN = 8;                    //  habilito modo RAPIDO
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estado
 ////////////////////////////////////////////////////////////

  j=0; i=0;
  j_SE=0; i_SE=0;

  data=0; j_out=0; i_out=0; opi_out=0; opj_out=0;

  j_im=0; i_im=0;

  printf("\n *** MORFOVAR. Procesamiento Morfológico ***");

  dila=0;           ero=0;
  dila_fail=0;      ero_fail=0;
  ya=0;

 while (i_im < heigth+1)
 {
// Bloques de analisis
   if (i_SE<3)  //Secuencia de entrada del SE
    {
       fread(&data_in,sizeof(char),1,fp_SE); //sizeof(char)=1 mide el tamaño de un char
       Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado*/

              if (j_SE==2) {j_SE=0; i_SE++;} else j_SE++;
    }
   else
    {       if ( i<heigth+1 )//& preparado==1 )
```

```
            {
             fread(&data_in,sizeof(char),1,fp_A);
             Write(BaseAddress,PIXEL,data_in); // Escribe en el Registro de estado

             if (j==width) {j=0; i++;} else j++;
            }

            if (i==2 & j==2)            /***** Filas necesarias listas *****/
             {ya=1;}

           if ( (ya == 1) & (i_im < heigth +1) )
            {
             Read(BaseAddress,DILA,dila); // Leo el resultados
             Read(BaseAddress,ERO,ero);

             dila= dila & 0x00FF;
             ero = ero & 0x00FF;

             fwrite(&dila,1,1,fp_DILA);
             fwrite(&ero,1,1,fp_ERO);

             if (j_im==width) {j_im=0; i_im++;} else j_im++;
            }
   }
}
gettime(&t2);

B_IN = 4; // Activo el PIN de RESET
Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

fclose(fp_SE);
fclose(fp_A);
fclose(fp_DILA);
fclose(fp_ERO);
  gotoxy(1,23);
  printf("Hora: Inicio=%2d:%02d:%02d      Finalizacion=%2d:%02d:%02d",
  t1.ti_hour, t1.ti_min, t1.ti_sec, t2.ti_hour, t2.ti_min, t2.ti_sec);


// *** BLOQUE DE OBTENCION DE RESULTADOS ***

   // *** VERIFICACION DE ERRORES de la DILACION ***
   fp_mdil = fopen("M_DILA.hex", "rb");
   fp_DILA = fopen("DILA.hex", "rb");

   while( (!feof(fp_mdil)) & (dila_fail==0) )
   {
     fread(&mo_dila,1,1,fp_mdil);
     fread(&ana_dila,1,1,fp_DILA);

     if (mo_dila!=ana_dila) dila_fail=1; else dila_fail=0;
   }
   fclose(fp_mdil);
   fclose(fp_DILA);

   // *** VERIFICACION DE ERRORES de la EROSION ***
```

```
    fp_mero = fopen("M_ERO.hex", "rb");
    fp_ERO = fopen("ERO.hex", "rb");

    while( (!feof(fp_mero)) & (ero_fail==0) )
    {
      fread(&mo_ero,1,1,fp_mero);
      fread(&ana_ero,1,1,fp_ERO);

      if (mo_ero!=ana_ero) ero_fail=1; else dila_fail=0;
     }
    fclose(fp_mero);
    fclose(fp_ERO);

    // Escribo valores en el archivo de texto
    fprintf(stream, "\n%d      ", num+1);    //numero de prueba

    if (dila_fail==0)
      fwrite(ok, strlen(ok)+1,1,stream);
    else
    { fwrite(falla, strlen(falla)+1,1,stream);
      n_fallas_dila++; }
      fprintf(stream,"      ");                    // Tiempo de duración

    if (ero_fail==0)
      fwrite(ok, strlen(ok)+1,1,stream);
    else
      { fwrite(falla, strlen(falla)+1,1,stream);
            n_fallas_ero++; }
/*************************************************************/

 B_IN = 4;//  RESET
 Write(BaseAddress,BANDERA_IN,B_IN); // Escribe en el Registro de estads

// FIN DEL BUCLE DE PRUEBAS

demora=1000*(3600*(t2.ti_hour-t1.ti_hour)+60*(t2.ti_min - t1.ti_min)+(t2.ti_sec -
t1.ti_sec))+10*(t2.ti_hund-t1.ti_hund);

 t_promedio=demora/max;

/******** FIN DEL PROCESO *******/

 fprintf(stream,"\n\n\n*** TIEMPO DE PROCESAMIENTO PROMEDIO= %.3f",t_promedio);

 fprintf(stream,"\n\n  HORA DE FINALIZACION :%2d:%02d",t2.ti_hour, t2.ti_min);

 fprintf(stream,"\n\n Numero de fallas en Dilasion = %d",n_fallas_dila);
 fprintf(stream,"\n Numero de fallas en Erosion = %d",n_fallas_ero);

 fprintf(stream,"\n\n ********** FIN DE REPORTE **********");
 fclose(stream); //cierro archivo reportes

 printf("\n Dilación y Erosión Procesadas");
 getch();
 return 0;
}
```

B.9. LIBRERÍA BMP_LIB.H

```
int bmp_hex(char bmp_name[20]);

int hex_bmp(char hex_name[20], char bmp_name[20]);

unsigned int ancho, alto;
unsigned int h=0, w=0, an=0;

unsigned char i; //esta es la variable que uso para leer y escribir
unsigned int cont=0;

/*********************************************************************/

int bmp_hex(char bmp_name[20])
{
 FILE *hexa;
 FILE *data;
 FILE *header;
 FILE *archivo;

 int numbits;

 char dato[2];

 if ( (archivo = fopen(bmp_name,"rb")) == NULL) return 1;

/*** Analiza Cabecera ***/

 fseek(archivo,0L, SEEK_SET);          //se lee y escribe la cabecera en el archivo head.hed

 fread(&dato, 1, 2, archivo);

 if (dato[0]!='B' | dato[1]!='M') return 2;   //Sale si archivo no es BMP

 fseek(archivo,18, SEEK_SET);
 fread(&ancho, 1, 4,  archivo);          //Leo el ancho de la imagen
 fread(&alto, 1, 4, archivo);            //Leo la altura de la imagen
 fread(&i, 1, 2, archivo);
 fread(&numbits, 1, 2 , archivo);        //Leo el numero de bits por pixel
 if ( i!=1 | numbits!=8 ) return 3;
 //Sale si la imagen no es en escala de grises

 if ( (ancho>256 | ancho<1) | (alto>256 | alto<1) ) return 4;
 //Sale si la imagen supera el tamaño m¡nimo

/*** EXTRACCION DE CABEZERA ***/
 header = fopen("head.hed","wb");

 fseek(archivo,0L,SEEK_SET);   //comenzamos a copiar los datos
 fseek(header,0L,SEEK_SET);    //comenzamos a copiar los datos

 for(cont=0; cont<1078; cont++)
  {
```

```
  fread(&i, 1 , 1, archivo);
  fwrite(&i, 1, 1, header);
 }

 fclose(header);

/**** EXTRACCION DE LA DATA ****/

 hexa = fopen("A.hex","wb");      //guardamos en el nuevo archivo hexa los datos sin ceros

 fseek(archivo,1078,SEEK_SET);          //comenzamos a copiar los datos

 if ( (ancho%4)==0 ) an=ancho;          //numero de ceros con los que se completa
 else an=ancho + 4 - (ancho) % 4;

 ancho--;
 alto--;

 fwrite(&ancho,1,1,hexa);          //escribo el ancho del archivo
 fwrite(&alto,1,1,hexa);           //escribo el ancho del archivo

 ancho++;
 alto++;
 w=0; h=0;

 while (h<alto)
 {
  if (w<ancho)                     //si el contador de columnas es menor q ancho entonces
   {
    fread(&i,1,1,archivo);         //leo el byte y lo guardo en una variable
    fwrite(&i,1,1,hexa);           //escribo la variable en el nuevo archivo
    w++;
   }
  else
   {
    if(w==an) {w=0; h++;}          //si llega al final entonces hace cero el contador
     else { fread(&i,1,1,archivo); w++; }          //solo lee pero no escribe
   }
 }

 fclose(hexa);
 fclose(archivo);

return 0;
}

/*****     Conversion de HEX a BMP     *****/
int hex_bmp(char hex_name[20], char bmp_name[20])
{
 FILE *header;
 FILE *abmp;
 FILE *data;

 if ( (header = fopen("head.hed","rb")) == NULL) return 1;

 if ( (abmp = fopen(bmp_name, "wb")) == NULL) return 2;
```

```
// Escritura de cabezera
 for(cont=0; cont<1078; cont++)
  {
   fread(&i,1,1,header);
   fwrite(&i,1,1,abmp);
  }
 fclose(header);
//////////////////////

 h=0; w=0;

 if ( (data = fopen(hex_name, "rb")) == NULL ) return 3;

 fseek(abmp,1078,SEEK_SET);          //comenzamos a copiar los datos
 fseek(data,0L,SEEK_SET);            //comenzamos a copiar los datos

 fread(&ancho,1,1,data);             //leo el ancho
 fread(&alto,1,1,data);              //leo el alto

 ancho++;
 alto++;

 if ((ancho%4)==0) an=ancho;         //numero de ceros con los que se completa
 else an=ancho + 4 - (ancho) % 4;

// Escritura de la DATA//
 while (h<alto)
 {
  if (w<ancho)
   {
    fread(&i,1,1,data);              //leo el byte y lo guardo en una variable
    fwrite(&i,1,1,abmp);             //escribo la variable en el nuevo archivo
    w++;
   }
  else
   {
    if(w==an) {w=0; h++;}            //si llega al final entonces hace cero el contador
     else {i=0; fwrite(&i,1,1,abmp); w++;}                    //Completo filas con ceros
   }
 }

fclose(data);
fclose(abmp);

return 0;
}
```

ANEXO C: CÓDIGO DEL PROGRAMA IMPLEMENTADO EN MATLAB PARA LA COMPROBACIÓN DE RESULTADOS

C.1. ALGORITMO DILACIÓN

DILA3.M

```
%Hace Dilacion para imagenes de cualquier tamaño
%con un Elemento estructural de 3x3

function [F,tiempo]=dila3(A,se)

[iAm jAm]=size(A);
T=zeros(3,3);
B=zeros(size(A));

tic;
for iA=1:iAm
  for jA=1:jAm

     for is=1:3
       for js=1:3

          iT=iA-is+2;
          jT=jA-js+2;
          if ( (iT>0 & iT<=iAm) & (jT>0 & jT<=jAm) )
             if (se(is,js)==0 | A(iT,jT)==0)
                T(is,js)=uint8(0);
             else
                T(is,js)=uint8(  double( A(iT,jT) ) + double( se(is,js) )  );
             end
          else T(is,js)=uint8(0);
          end

       end
     end

     B(iA,jA)=max(max(T));
  end
end
tiempo=toc;
F=uint8(B);
```

C.2. ALGORITMO EROSIÓN

ERO3.M

```
%Hace Erosion para imagenes de cualquier tamaño
%con un Elemento estructural de 3x3

function [F,tiempo]=ero3(A,se)

[iAm jAm]=size(A);
T=zeros(3,3);
B=zeros(size(A));

tic;
for iA=1:iAm
  for jA=1:jAm

     for is=1:3
       for js=1:3

          iT=iA-is+2;
          jT=jA-js+2;

          if ( (iT>0 & iT<=iAm) & (jT>0 & jT<=jAm) )
              T(is,js) = A(iT,jT);
          else T(is,js)=uint8(0);
          end

          if ( se(is,js)==0)
            T(is,js)=uint8(255);
          else
            T(is,js)=uint8(  double( T(is,js) ) - double( se(is,js) )  );
          end

       end
     end

     B(iA,jA)=min(min(T));
  end
end
tiempo=toc;
F=uint8(B);
```

C.3. PROGRAMA DE MEDICIÓN DE TIEMPOS DE PROCESAMIENTO

TEST.M

```
%Rutina para medir tiempos
clear all;
close all;
clc;

L=8;    %Defino tamaño inicial
B=[1 1 1; 1 1 1; 1 1 1]; % Elemento Estructural de 3x3

fprintf('Cuantas pruebas quiere realizar por tamaño:');
max=input(' ');

while (L <= 256)

   t_dila=0;
   t_ero=0;
   T_total_Dila=0;
   T_total_Ero=0;


   fprintf('\n\n***** Prueba para imagen de tamaño %d x %d *****\n\n',L,L);

   for i=1:max
      A=10*ones(L,L);
      B=zeros(L,L);
      [B,t_dila]=dila3(A,B);
      [B,t_ero]=ero3(A,B);

      T_total_Dila = T_total_Dila + t_dila;
      T_total_Ero = T_total_Ero + t_ero;
   end

   T_total_Dila = T_total_Dila / max;
   T_total_Ero = T_total_Ero / max;

   fprintf('\n  Tiempo promedio para DILACION: %d \n',T_total_Dila);
   fprintf('\n  Tiempo promedio para EROSION: %d \n',T_total_Ero);

   L=L*2;
end
```