PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

COPROCESADOR MORFOLÓGICO EMBEBIDO USANDO ARREGLOS SISTÓLICOS Y CON DESCRIPCIÓN DE HARDWARE PORTABLE

Tesis para optar por el Título de **Ingeniero Electrónico**, que presenta el bachiller:

Mario Alejandro Valega Pacora

ASESOR: Ph.D. Gerard Franz Santillán Quiñonez

Lima, julio de 2013

Resumen

Las operaciones morfológicas son ampliamente usadas en diversas aplicaciones relacionadas con el procesamiento digital de imágenes, permitiendo, entre otras aplicaciones, el reconocimiento de objetos, segmentación de imágenes, análisis de texturas y extracción de regiones. Este trabajo presenta el diseño, la descripción de hardware y caracterización de un coprocesador morfológico capaz de ejecutar las operaciones morfológicas de dilatación, erosión, apertura, cerradura, gradiente interno y gradiente externo. Este coprocesador opera sobre imágenes binarias con una rapidez de procesamiento apropiada. El trabajo parte de los conceptos de operaciones unidimensionales y bidimensionales para llegar a formular una arquitectura basada en arreglos sistólicos que permite el procesamiento de varias filas y columnas de una imagen de manera simultánea. El diseño de este coprocesador es parametrizable y portable, siendo posible variar el número de procesadores elementales que componen el arreglo sistólico y sintetizar el circuito para cualquier tecnología disponible. Además, el diseño incluye también la interfaz de comunicación entre un procesador y el coprocesador. Finalmente, se presentan resultados de la síntesis del circuito, así como su consumo de recursos y rapidez de procesamiento para diferentes configuraciones y plataformas.

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título	: Coprocesador Morfológico Embebido Usando Arreglos Sistólicos y con Descripción de Hardware Portable
Área	: Circuitos y Sistemas 1107
Asesor	: Gerard Franz Santillán Quiñonez
Alumno	: Mario Alejandro Valega Pacora
Código	: 20084418
Fecha	: 24/04/13

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
20 JUN. 2013
DANIEL TORREALVA DAVILA
DECANO

Descripción y Objetivos

Siendo la morfología matemática una metodología usada en diferentes campos de aplicación, el desarrollo de un coprocesador con operadores morfológicos fisicamente realizados por circuitos digitales puede incrementar la eficiencia del procesamiento digital de señales realizados en hardware. Así, el objetivo de esta tesis es desarrollar un coprocesador morfológico embebido con interfaces apropiadas para interactuar de forma eficiente con un procesador. La arquitectura propuesta debe ser descrita en un lenguaje de descripción de hardware en una forma tal que pueda ser sintetizado para cualquier tecnología disponible en una plataforma de desarrollo que soporte el lenguaje de descripción de hardware usado. A continuación, se muestran los objetivos específicos que se desean alcanzar:

- Diseñar en VHDL cada uno de los módulos que conforman el coprocesador morfológico e implementarlos en un FPGA.
- Proponer una estrategia apropiada para dividir los procesos y algoritmos para poder implementar los arreglos sistólicos de cada operación morfológica con el fin de incrementar la rapidez de procesamiento sin generar errores al hacer los cálculos.
- Optimizar el circuito para usar la menor cantidad de elementos lógicos programables del FPGA y, de esta forma, reducir el consumo de potencia.
- Realizar simulaciones usando las herramientas adecuadas para observar la respuesta del circuito ante diferentes estímulos.
- Verificar experimentalmente el correcto funcionamiento del sistema propuesto en una tarjeta de desarrollo basado en FPGAs.

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
SECCIÓN ELECTRICIDAD Y ELECTRÓNICA
Dr. Ing. BENJAMÍN CASTAÑEDA APHAN
Coordinador de la Especialidad de Ingeniería Electrónica

MÁXIMO 50 PÁGINAS

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título : Coprocesador Morfológico Embebido Usando Arreglos Sistólicos y con Descripción de Hardware Portable

Índice

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
SECCIÓN ELECTRICIDAD Y ELECTRÓNICA

Dr. Ing. BENJAMÍN CASTAÑEDA APHAN
Coordinador de la Especialidad de Ingeniería Electrónica

MÁXIMO 50 PÁGINAS

Índice

Lista de Figuras

Lista de Tablas

Lista de Acrónimos

CPLD : Complex Programmable Logic Device

FPGA : Field Programmable Gate Arrays

SE : Structural Element

VHDL : Very High Speed Integrated Circuit Hardware Description Language

Introducción

Las operaciones morfológicas son ampliamente usadas en diversas aplicaciones relacionadas con el procesamiento digital de imágenes, permitiendo, entre otras aplicaciones, el reconocimiento de objetos, segmentación de imágenes, análisis de texturas y extracción de regiones [8] [16]. En general, estas operaciones se basan en cálculos que deben realizarse sobre cada uno de los píxeles de una imagen, lo cual representa un alto costo computacional [9]. Sin embargo, los algoritmos de procesamiento morfológico son inherentemente paralelos, por lo cual, cualquier implementación de estas operaciones se podría beneficiar al partir de un esquema de computación paralela [8]. Partiendo de esta afirmación, la implementación de un sistema dedicado al desarrollo de operaciones morfológicas con un grado de paralelismo apropiado podría incrementar la rapidez de procesamiento.

Por otro lado, son varias las metodologías o arquitecturas de *hardware* elaboradas para sistemas que realicen operaciones en paralelo. Dentro de esta gran variedad, destaca un tipo conocido como arreglo sistólico, que surgió como una forma de llevar los cálculos de alto nivel a estructuras de hardware de una manera ordenada y eficiente [6].

A medida que el uso de sistemas embebidos se ha incrementado de manera exponencial [2], las características de los dispositvos programbles como los FPGAs también han mejorado, permitiendo el desarrollo de circuitos cada vez más complejos y a la vez más eficientes. Además, la inclusión de *soft-processors* en estos dispositivos brinda la posibilidad de desarrollar un sistema completo combinando software con hardware y dando como resultado un *System on a Programmable Chip*, SOPC.

Dado este contexto, este trabajo tiene como objetivo principal diseñar un coprocesador morfológico capaz de realizar las operaciones morfológicas básicas con una rapidez apropiada y cuya arquitectura esté basada en arreglos sistólicos. La descripción de este sistema debe ser portable y parametrizable, brindando así la posibilidad de ser sintetizado para cualquier plataforma basada en FPGAs. Aparte del desarrollo del coprocesador, este trabajo incluye el diseño de la interfaz adecuada para comunicarse con el procesador, elaborando así un sistema SOPC completo, donde el procesador se encarga del control, envío y recepción de datos, mientras que el coprocesador morfológico sirve como acelerador para las operaciones morfológicas.

De acuerdo al objetivo general planteado, el trabajo se ha estructurado como sigue. En el Capítulo 1, se muestra el enfoque de trabajo que se ha seguido y cuáles con los objetivos específicos que se busca alcanzar con este trabajo. Así mismo, se muestra el estado del arte correspondiente a los diversos trabajos sobre morfología realizados mediante el diseño de hardware específico. Por último, se explica de manera breve y concreta el concepto de morfología matemática y sus dos operadores básicos, así como los conceptos de sistema embebido y portabilidad. En el Capítulo 2, se parte de la definición de operación morfológica unidimensional y bidimensional para llegar al planteamineto de un arreglo sistólico adecuado. También, se indica de manera general la forma de comunicación entre el procesador y el coprocesador. En el Capítulo 3, se explica de manera detallada cada uno de los módulos que componen el coprocesador morfológico, partiendo de los módulos elementales hasta llegar al diseño completo de la unidad morfológica que abarca el arreglo sistólico y su respectiva unidad de control. En el Capítulo 4, se muestran los resultados obtenidos tanto para el consumo de recursos como para la rapidez de procesamiento del coprocesador. Además, se explica la metodología seguida para realizar las pruebas y se muestran los resultados de simulación. Por último, el trabajo termina con la presentación de las conclusiones y recomendaciones en base a los resultados obtenidos.

Capítulo 1

Fundamentación

En este capítulo, se presenta el estado del arte relacionado con los diversos trabajos realizados sobre morfología matemática en *hardware*, tanto programable como de aplicación específica. Así mismo, se muestran los objetivos específicos que se buscan alcanzar con este trabajo. Por último, se explican algunos conceptos teóricos fundamentales en el desarrollo de este trabajo.

1.1. Estado del arte

1.1.1. Presentación del asunto de estudio

La morfología matemática es una teoría muy útil en el procesamiento digital de imágenes. Las operaciones morfológicas pueden simplificar los datos de una imagen, preservar sus características esenciales y eliminar los aspectos irrelevantes [13]. Sus aplicaciones más recientes están relacionadas con la resolución de problemas geográficos, análisis de formas tridimensionales, detección de nódulos de grafito en fundición dúctil y análisis de imágenes médicas [7] [12]. Por otro lado, estas operaciones implican un alto costo computacional [10]. Así, diversos trabajos plantean el uso de sistemas especialmente desarrollados en *hardware* para realizar estas operaciones, aprovechando además la capacidad inherente de los algoritmos morfológicos para ser paralelizados. Siguiendo este enfoque, este trabajo busca estudiar y analizar una arquitectura conocida como arreglos sistólicos para llegar a formular y diseñar un coprocesador que permita realizar las operaciones morfológicas básicas sobre imágenes binarias.

1.1.2. Estado de la investigación

En [17] se implementaron las operaciones morfológicas de dilatación y erosión en *hardware* usando un CPLD, cuyos elementos lógicos no eran suficientes para implementar di-

rectamente los algoritmos. Por este motivo, se dividió el algoritmo en partes más pequeñas usando el concepto de morfología unidimensional y bidimensional. De esta forma, se consiguió usar una menor cantidad de recursos, desarrollando un módulo más pequeño que realiza operaciones parciales de manera iterativa para finalmente unir todos los resultados parciales y obtener la imagen ya procesada. El problema con este diseño es la ausencia de registros internos para almacenar resultados parciales y, dada la gran cantidad de iteraciones necesarias, el constante acceso a memoria reduce la eficiencia del sistema.

Otro método también aplicable para la implementación en *hardware* es el que se basa en operadores de ventana. El centro de técnicas analógicas y digitales de la UNLP de Argentina implementó algoritmos de morfología matemática en un FPGA basándose en este concepto [11]. Este método aprovecha que muchas de las operaciones de procesamiento de imágenes se basan en operadores de ventana para desarrollar bloques comunes para todas las operaciones. Se distinguen tres etapas principales. Una etapa de generación de ventana, un contador de filas y columnas y la etapa final específica del algoritmo a implementar. Las dos primeras etapas pueden implementarse en dos bloques útiles para todas las operaciones. De esta forma, la cantidad de recursos usados es reducida sin sacrificar de manera considerable la rapidez de procesamiento.

A diferencia de los trabajos anteriores donde se busca disminuir el consumo de recursos, [5] busca incrementar la rapidez de procesamiento a través de la implementación en hardware de un arreglo sistólico unidimensional, capaz de realizar operaciones morfológicas sobre imágenes binarias y en escala de grises, mediante el empleo de varios procesadores elementales llamados ESP (Envelope Scan Processor).

Otro trabajo analizado es el realizado por el departamento de electrónica de la Universidad de Liverpool, en el cual se desarrolla un sistema morfológico mediante el uso de la herramienta de desarrollo *DSP Builder*, proporcionado en el software *Simulink* de *Matlab* [14]. Esta herramienta permite la elaboración de un diseño a nivel gráfico que luego es convertido a una descripción en VHDL. El diseño se basa en buffers que se encargan de dividir las imágenes de entrada en segmentos de 3x3, los cuales se envían de manera secuencial a un *calculador morfológico*. La implementación de este calculador consiste en un arreglo en cascada de compuertas *AND* y *OR*. La limitación de esta arquitectura es que sólo permite realizar las operaciones de dilatación y erosión para un elemento estructural de 3x3 en configuración *4-connectivity* o *8-connectivity*. Además, como sólo se halla el valor de un píxel de la imagen por operación, la ganancia en rapidez es mínima.

En [15] se analiza el uso de elementos estructurales de diferentes formas que permiten el desarrollo de algoritmos más sofisticados. Siguiendo esta línea, se muestra que la implementación de estas operaciones morfológicas se puede conseguir usando sólo lógica booleana y elementos de retardo, lo cual permite una implementación directa en los FPGAs.

Otra metodología para trabajar con las operaciones morfológicas consiste en formar nuevos elementos estructurales mediante descomposiciones recursivas [4]. Así, el trabajo analizado usa una arquitectura totalmente regular y permite realizar las operaciones de dilatación y erosión en un solo paso de la imagen, sin importar el tamaño del elemento estructural.

Las implementaciones en hardware no sólo abarcan diseños digitales, sino que otros trabajos como el elaborado por el Georgia Institute of Technology proponen la implementación de un arreglo analógico mediante el uso de fotodetectores, permitiendo así la entrada de datos en paralelo de manera óptica. [8].

Se observa entonces que el desarrollo de operadores morfológicos es un tema de interés de los investigadores, quienes han desarrollado una gran cantidad de metodologías para descomponer los algoritmos y poder diseñar arquitecturas eficientes. Se ha visto también, que los desarrollos abarcan tanto el ámbito digital como analógico y los dispositivos usados son tanto programables, como FPGAs y CPLDs, como de aplicación específica (VLSI).

1.2. Objetivos

1.2.1. Objetivo General

El objetivo general de este trabajo consiste en diseñar un coprocesador morfológico embebido con las interfaces apropiadas para interactuar de manera eficiente con un procesador, descrito de tal forma que sea portable y sintetizable para cualquier tecnología disponible.

1.2.2. Objetivos Específicos

- Proponer una estrategia adecuada para dividir los procesos y algoritmos de cada operador morfológico, de tal forma que puedan implementarse en un arreglo sistólico con el fin de incrementar la rapidez de procesamiento.
- Diseñar y describir en VHDL cada uno de los módulos que conforman el coprocesador morfológico, de forma tal que la descripción sea portable.
- Optimizar el circuito para usar la menor cantidad de elementos lógicos programables del FPGA y, de esta forma, reducir el consumo de potencia.
- Realizar simulaciones usando herramientas adecuadas como Isim y RTL Viewer de los entornos de desarrollo ISE y Quartus II para observar la respuesta del circuito ante diferentes estímulos.

1.3. Morfología Matemática

La morfología matemática se puede definir como el análisis de estructuras espaciales. Se le conoce como morfología porque tiene como objetivo el análisis de las formas de los objetos. Además, se le conoce como matemática debido a que se basa en la teoría de conjuntos [13]. Dentro de la morfología matemática, los operadores de dilatación y erosión son los básicos y a partir de éstos se pueden elaborar otros más complejos. A continuación, se presenta la definición de ambos operadores para el caso de imágenes binarias.

1.3.1. Dilatación

En general, los operadores morfológicos son operadores locales o de vecindad, ya que el resultado para cada píxel es definido por los píxeles que lo rodean, de acuerdo a una vecindad que es definida por un elemento estructural. El elemento estructural es otro conjunto de forma conocida, que es seleccionado de acuerdo a un conocimiento previo de la geometría de las estructuras relevantes e irrelevantes de la imagen que se desea operar. En la Ecuación 1.1 se muestra la definición matemática de la dilatación, donde X representa el conjunto corrspondiente a la imagen y B el elemento estructural.

$$\delta_B(X) = \{x |\, B_x \cap X \neq 0\} \tag{1.1}$$

La interpretación de la Ecuación 1.1 consiste en ubicar el origen del elemento estructural en cada uno de los píxeles de la imagen y verificar si el elemento estructural intercepta a la imagen [13]. El resultado de esta transformación es el conjunto de puntos que cumplen con esta condición. Un ejemplo de dilatación se muestra en la Figura 1.1.

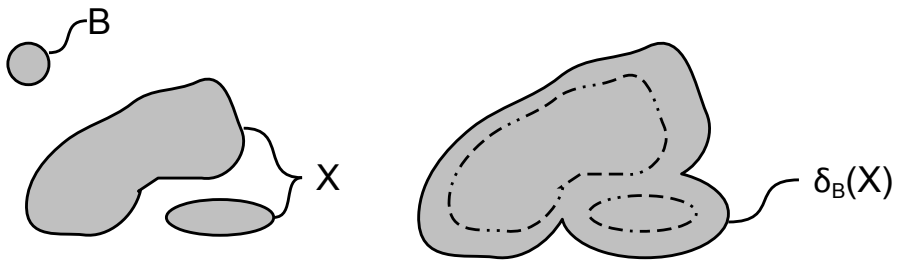


Figura 1.1: Ejemplo de dilatación para un elemento estructural con forma de disco.

1.3.2. Erosión

Al igual que la dilatación, la erosión consiste en desplazar un elemento estructural por cada píxel de la imagen. En este caso, la condición que se debe cumplir es que el elemento estructural esté contenido en la imagen [13]. Un ejemplo de erosión se muestra en la Figura 1.2. La definición matemática de este operador se muestra en la ecuación 1.2.

$$\varepsilon_B(X) = \{x \,|B_x \subseteq X\} \tag{1.2}$$

Es importante mencionar que existe una relación entre este operador y la dilatación, conocida como dualidad. De acuerdo a esta propiedad, la erosión es equivalente al complemento de la dilatación del complemento de la imagen. Esto se muestra en la Ecuación 1.3.

$$\varepsilon_B(X) = (\delta_B(X^c))^c \tag{1.3}$$

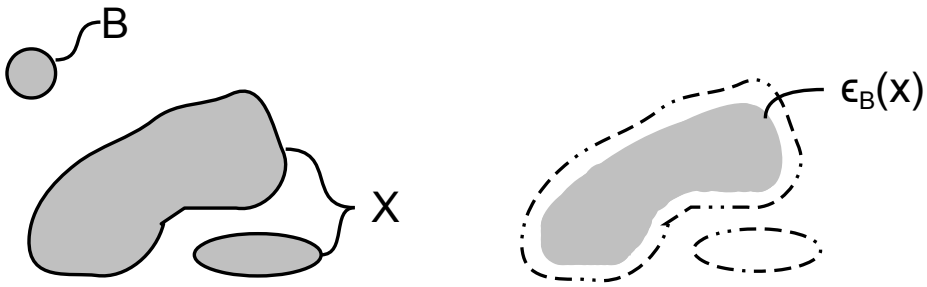


Figura 1.2: Ejemplo de erosión para un elemento estructural con forma de disco.

1.3.3. Apertura

La apertura de una imagen f por un elemento estructural B consiste en la erosión de f por B seguida de la dilatación por la transpuesta de B.

$$\gamma_B(f) = \delta_{\breve{B}}[\varepsilon_B(f)] \tag{1.4}$$

Es conveniente mencionar que cuando el elemento estructural B es simétrico, entonces su transpuesta será igual a B. En general, no existe transformación inversa para la erosión que permita recuperar la imagen original. Sin embargo, la idea de la apertura es dilatar

la imagen erosionada para recuperar lo más que se pueda la imagen original. La apertura permite alisar contornos, eliminar las protuberancias donde no quepa el elemento estructural y separar objetos en puntos estrechos.

1.3.4. Cerradura

Esta operación tiende a recuperar la forma inicial de las estructuras de una imagen que ha sido dilatada. La cerradura de una imagen f por un elemento estructural B se define de la siguiente forma:

$$\phi_B(f) = \varepsilon_{\breve{B}}[\delta_B(f)] \tag{1.5}$$

Esta operación tiende a alisar porciones del contorno, fusionar estrechas grietas y rellenar agujeros pequeños.

1.3.5. Gradiente interno

El gradiente interno ρ^-, también conocido como medio gradiente por erosión, se define como la diferencia entre la imagen original y la imagen erosionada:

$$\rho_B{}^- = X - \varepsilon_B(X) = X - [\delta_B(X^c)]^c = X \cap \{[\delta_B(X^c)]^c\}^c = X \cap \delta_B(X^c) \tag{1.6}$$

Esta operación solo está definida para elementos estructurales simétricos. En imágenes binarias, permite obtener una máscara de los bordes internos de los objetos de la imagen.

1.3.6. Gradiente externo

El gradiente externo ρ^+, también conocido como medio gradiente por dilatación, se define como la diferencia entre la imagen dilatada y la imagen original:

$$\rho_B{}^+ = \delta_B(X) - X = \delta_B(X) \cap X^c \tag{1.7}$$

Al igual que en el gradiente interno, está operación sólo está definida para SE simétricos. En imágenes binarias, permite obtener una máscara de los bordes externos de los objetos de la imagen.

1.4. Operaciones Unidimensionales y Bidimensionales

En esta sección se explican los conceptos de dilatación y erosión unidimensional y bidimensional, que son fundamentales en el diseño del arreglo sistólico.

1.4.1. Dilatación y Erosión Unidimensional

La dilatación y erosión unidimensional consisten en realizar estas operaciones de acuerdo a lo señalado en el capítulo anterior, pero limitándolas a una sola fila, tanto de la imagen como del elemento estructural. Así, se desplaza una fila del elemento estructural por cada píxel de una fila de la imagen. Para el caso de la dilatación, donde el píxel de la imagen sea "1", se coloca el elemento estructural, mientras que cuando el píxel vale "0", se colocan ceros en la vecindad delimitada por el elemento estructural. Finalmente, el resultado consiste en la unión de todos los elementos que se encuentran en una misma columna. En este caso de imágenes binarias, la unión es equivalente a aplicar la compuerta lógica *OR*. El procedimiento antes explicado lo muestra la Figura 1.3, donde la fila de la imagen tiene 6 píxeles y es representada por A, mientras que el elemento estructural es representado por B y tiene 3 píxeles. El resultado obtenido se muestra en la parte inferior y se ve que es la unión de todos los elementos de una misma columna que se obtienen como resultado de realizar los desplazamientos del elemento estructural. En este caso, el segundo y quinto píxel de la imagen valen 0, por lo cual en dichas posiciones, en lugar de colocar el elemento estructural, se colocan 0s. Finalmente, se debe observar que el resultado contiene ocho píxeles, es decir, dos píxeles adicionales. En general, al operar de esta forma, se obtendrá un número de pixeles adicionales igual al tamaño del elemento estructural menos uno. En el caso de elementos estructurales impares y cuyo origen coincida con su centro geométrico, se deben descartar los píxeles extremos. En este ejemplo, hay dos píxeles adicionales, por lo cual se descarta el primer y el último píxel, los cuales se muestran sombreados.

Para el caso de erosión unidimensional, se puede aplicar el concepto de dualidad mencionado en el capítulo anterior. Así, el procedimiento es similar al de la dilatación unidimensional, con la diferencia de que en este caso no se opera directamente sobre los píxeles de la imagen, sino sobre su complemento. Además, también se debe hallar el complemento del resultado. En este caso, como se está trabajando con imágenes binarias, el complemento es equivalente a una compuerta lógica *NOT*. En la Figura 1.4 se muestra un ejemplo de erosión para la misma fila de imagen y elemento estructural usados en el caso de la dilatación.

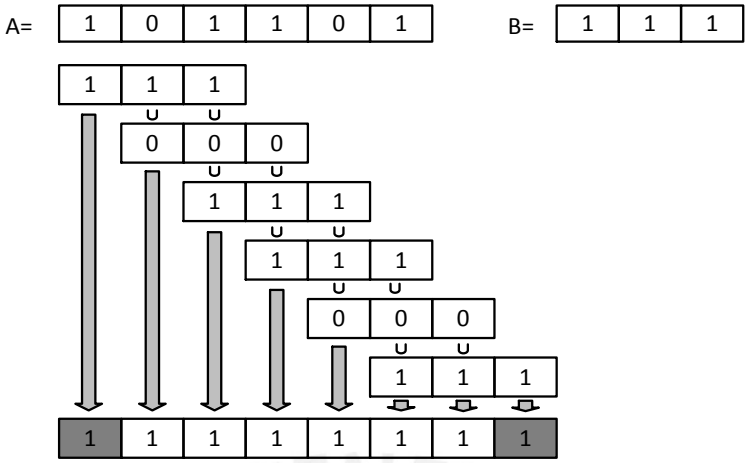


Figura 1.3: Dilatación unidimensional

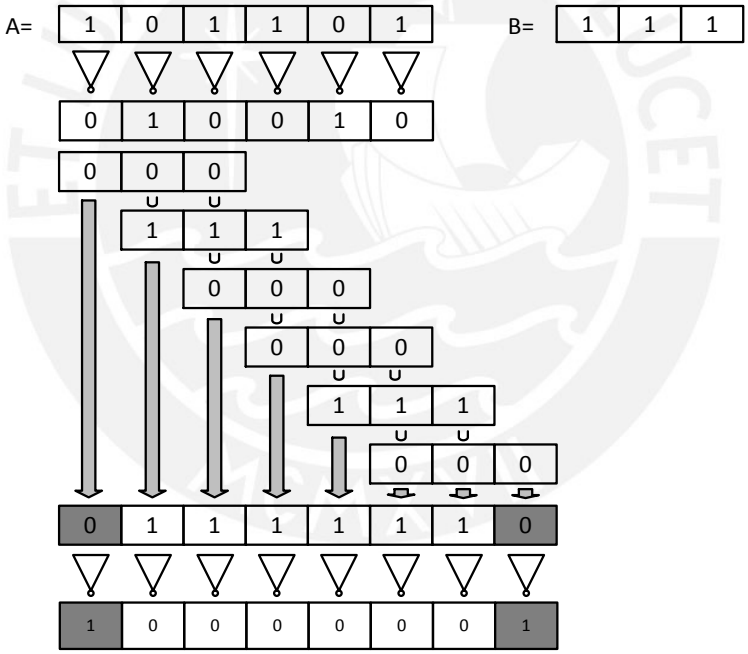


Figura 1.4: Erosión unidimensional

1.4.2. Dilatación y Erosión Bidimensional

En la sección anterior se han explicado los conceptos básicos de dilatación y erosión unidimensional. En esta sección se muestra cómo estos conceptos sirven como base para la construcción de la dilatación y erosión bidimensional, permitiendo así el procesamiento de una imagen completa.

La dilatación bidimensional consiste básicamente en realizar la dilatación unidimensional de cada una de las filas de la imagen con cada una de las filas del elemento estructural y unir los resultados que corresponden a la misma fila de la imagen transformada. De la misma forma, la erosión bidimensional consiste en realizar la erosión unidimensional de cada fila de la imagen con el elemento estructural e interceptar los resultados correspondientes a la misma fila de la imagen transformada. La Figura 1.5 muestra la operación bidimensional para una imagen de 16 filas con un elemento estructural de 9 filas. Cada bloque representa una operación unidimensional, ya sea dilatación o erosión, entre una fila de la imagen y una fila del elemento estructural. Se observa que la suma de la fila de la imagen con la fila del elemento estructural dentro de cada bloque coincide con la fila que ocupa en la imagen procesada. El resultado de cada fila de la imagen procesada es igual a la unión o intercepción de los resultados de todos los bloques ubicados en una misma fila, de acuerdo a si se realiza la dilatación o erosión, respectivamente. Al igual que con las operaciones unidimensionales, en este caso también se obtienen datos adiconales. En general, la imagen tendrá un número de filas adicionales equivalente al número de filas del elemento estructural menos uno. Considerando que el elemento estructural es impar y que su origen se encuentra en la fila central, se deben descartar las filas de los extremos. Por ejemplo, en el caso mostrado en la Figura 1.5 el elemento estructural tiene 9 filas, por lo cual se deben descartar 8 filas, que son las cuatro primeras y las cuatro últimas. Estas se muestran sombreadas.

1.5. Sistemas Embebidos

Los sistemas embebidos son actualmente empleados en una gran variedad de dispositivos como celulares, controles de aviones, dispositivos médicos, entre otros. Esta tendencia ha incrementado desde el año 1999, periodo en el cual, sólo el 1 % de los procesadores producidos fueron usados en computadoras de propósito general [3]. La gran mayoría fue utilizada en el mercado de sistemas embebidos.

Los sistemas embebidos se caracterizan por su función dedicada, comportamiento de tiempo real y cumplimiento de altos requerimientos. Estos sistemas incluyen elementos de software y hardware; esto es, procesadores programables y componentes de hardware como circuitos integrados de aplicación específica o programables como los FPGA's.

Todos estos componentes pueden ser incluidos en un solo circuito integrado, comúnmente llamado SoC (System on a Chip). En este trabajo, se implementará una variación del SoC, llamada SOPC(System on a Programmable Chip), por la capacidad para ser programado de un FPGA [2].

Cuando se diseña un sistema basado en un procesador embebido convencional, se examinan las funciones requeridas y luego se selecciona el procesador y los periféricos de entrada y

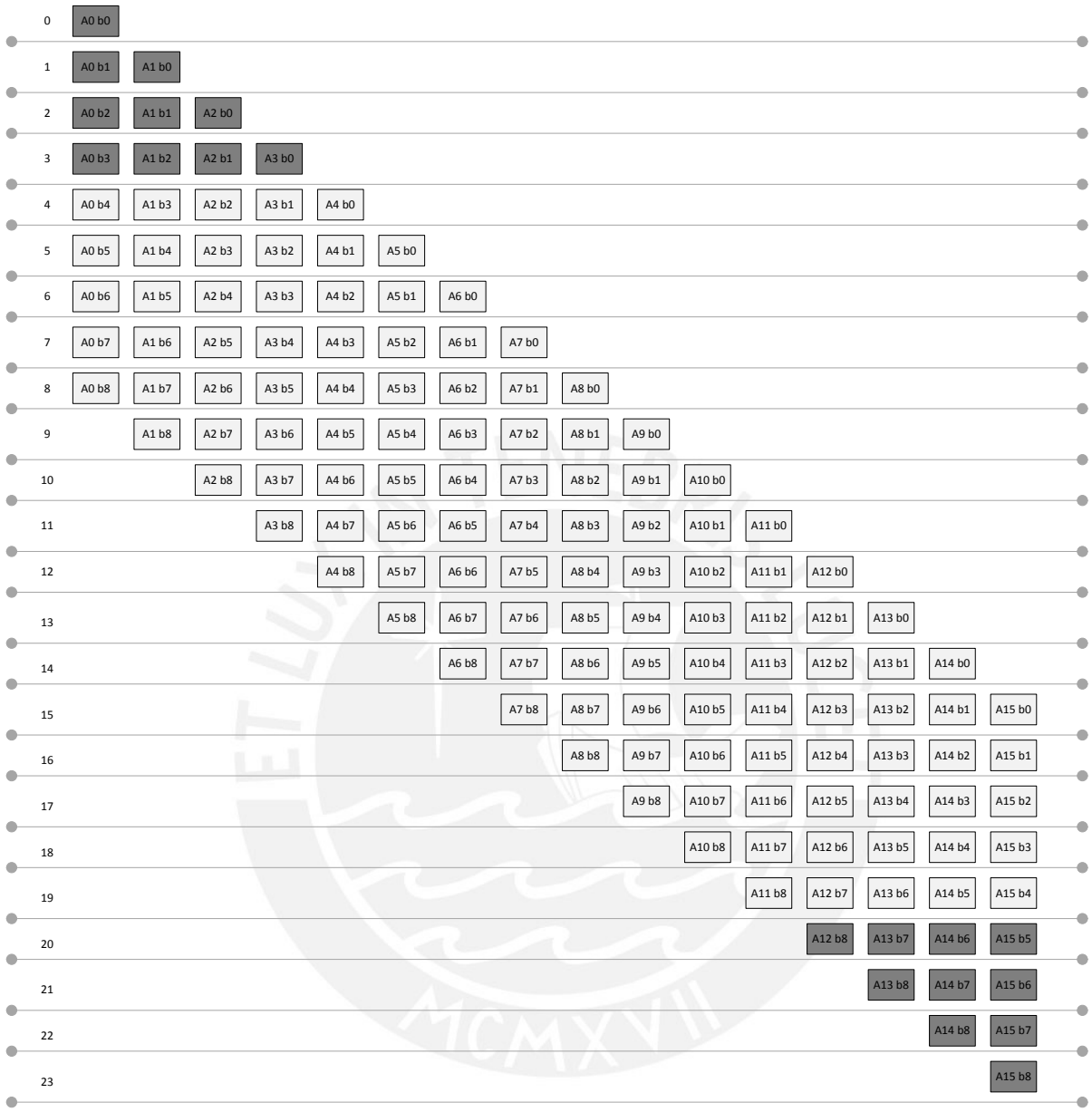


Figura 1.5: Dilatación bidimensional

salida para construir la plataforma de hadware. En cambio, un FPGA brinda celdas lógicas e interconexiones que pueden ser configuradas para realizar una función específica. Esta programabilidad de los dispositivos FPGA permite incorporar *hardware* personalizado que puede servir como acelerador para tareas de gran costo computacional.

1.6. Descripción de Hardware Portable

Por descripción portable se entiende que el código desarrollado debe poder ser sintetizado por cualquier software de desarrollo e implementado en cualquier dispositivo adecuado, lo cual implica no usar librerías de alguna plataforma específica, sino que se deben emplear sólo librerías estándares. En este sentido, es necesario investigar sobre las librerías a usar, ya que en algunos casos éstas han sido desarrolladas por compañías privadas, pero aún así han sido incluidas dentro de las librerías IEEE, dando la impresión de ser un estándar. Un ejemplo es la librería std_arith, la cual fue desarrollada por la empresa Synopsys, pero está incluida en la librería IEEE, a pesar de ocasionar problemas de incompatibilidad. Por otro lado, siguiendo un enfoque más moderno, no sólo es suficiente con que el circuito descrito pueda usarse en una variedad de plataformas, sino que además, debe poder adecuarse a los recursos del dispositivo y a las necesidades del usuario. Así, en este trabajo se ha realizado una descripción parametrizada en VHDL, haciendo uso de las sentencias *if-generate* y *for-generate*, propias del VHDL y orientadas al diseño de circuitos digitales más complejos a partir de módulos básicos repetitivos [1]. De esta forma, variando los valores de algunos parámetros, es posible cambiar las dimensiones del arreglo bidimensional que realiza las operaciones morfológicas. Así, se puede ajustar la arquitectura a los recursos disponibles en el dispositivo o a las especificaciones de la aplicación.

Capítulo 2

Sistema Propuesto

2.1. Arquitectura General del Sistema Propuesto

En esta sección se muestra de manera general la propuesta planteada para el diseño del coprocesador y su comunicación con el procesador. La Figura 2.1 muestra que la comunicación entre el procesador y el coprocesador se realiza mediante dos buses unidireccionales: a) *bus* A, por donde el procesador envía datos al coprocesador, y b) *bus* B por donde el procesador recibe los datos desde el coprocesador. Además, se aprecia que hay una interfaz de comunicación y control, la cual interactúa directamente con las señales de control de los buses. Así mismo, este módulo se encarga de decodificar las instrucciones mostradas en la Tabla 2.1 y de controlar los registros de entrada, la unidad morfológica y el archivo de registros. Los datos, tanto de la imagen como del elemento estructural, se encuentran almacenados en una memoria externa, a la cual accede el procesador. Estos datos son enviados mediante el *bus* A, junto con unas señales de control propias de este bus. La interfaz se encarga de leer y cargar cada uno de los datos recibidos a los registros de entrada. Desde estos registros los datos son cargados en otra estructura de almacenamiento conocida como archivo de registros, si es que éstos son parte del elemento estructural o directamente cargados dentro de la unidad morfológica, si éstos corresponden a una parte de la imagen. Así, la arquitectura planteada permite realizar operaciones sobre imágenes distintas usando un mismo elemento estructural, sin tener que cargarlo nuevamente. En la Sección 4.3 se explica el diseño de esta propuesta para un procesador y bus específicos.

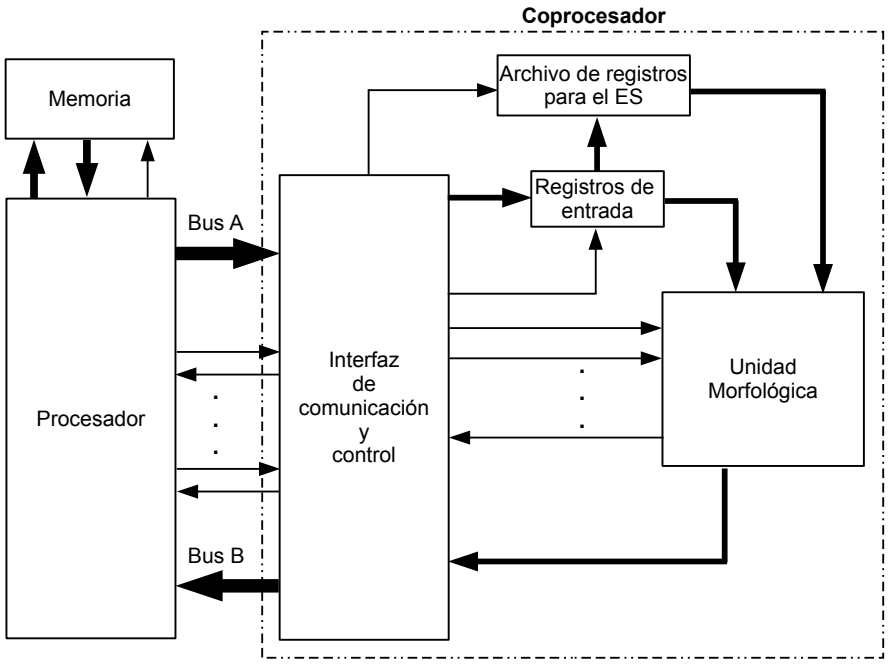


Figura 2.1: Propuesta de comunicación entre procesador y coprocesador

Tabla 2.1: Lista de instrucciones

Código	Función
001	Grabar elemento estructural
010	Iniciar operación de dilatación
011	Iniciar operación de erosión
100	Iniciar operación de apertura
101	Iniciar operación de cerradura
110	Iniciar operación de gradiente interno
111	Iniciar operación de gradiente externo

2.2. Arreglo Sistólico

En base al flujo de datos observado en el esquema bidimensional, se plantea un arreglo sistólico apropiado que permita la ejecución de varias operaciones unidimensionales de manera simultánea. En la Figura 1.5 se observa que la estructura de la operación bidimensional tiene una forma diagonal. Además, puede observarse que todos los bloques que se encuentran en una misma columna comparten la misma fila de la imagen, mientras que todos los bloques que se encuentran en la misma diagonal comparten la misma fila del elemento estructural. En base a estas observaciones se formula un arreglo sistólico que posea esta forma y cuyas interconexiones internas faciliten el flujo de datos que se ha descrito. Así, la Figura 2.2

muestra un esquema básico del arreglo sistólico propuesto para el desarrollo de las operaciones de dilatación y erosión. En este caso, hay 12 bloques independientes denominados PEU (Procesador Elemental Unidimensional), capaces cada uno de realizar la dilatación y erosión unidimensional. Se observa que, de acuerdo a lo mencionado anteriormente, los datos de la imagen se transfieren verticalmente, mientras que los datos del elemento estructural son transferidos de manera diagonal. Aparte de los bloques PEU, se incluyen registros A para almacenar las filas de las imágenes y registros B para almacenar las filas del elemento estructural. Cuando el número de filas del elemento estructural excede el número de bloques PEU por fila del arreglo sistólico, entonces se deberá realizar operaciones consecutivas con las siguientes filas del elemento estructural hasta cubrir todas. Por ejemplo, si se quisiera usar el arreglo sistólico de la Figura 2.2, que contiene 3 PEUs por fila, para realizar la operación bidimensional mostrada en la Figura 1.5, donde el elemento estructural tiene 9 filas, se necesitarían 3 iteraciones para obtener el resultado de las cuatro primeras filas de la imagen transformada. Así, se observa que a mayor número de PEUs en una misma fila del arreglo, menor es el número de iteraciones necesarias para obtener el resultado de una fila. Además, a mayor número de PEUs en una columna del arreglo, mayor es el número de filas de la imagen transformada que se pueden procesar en simultáneo.

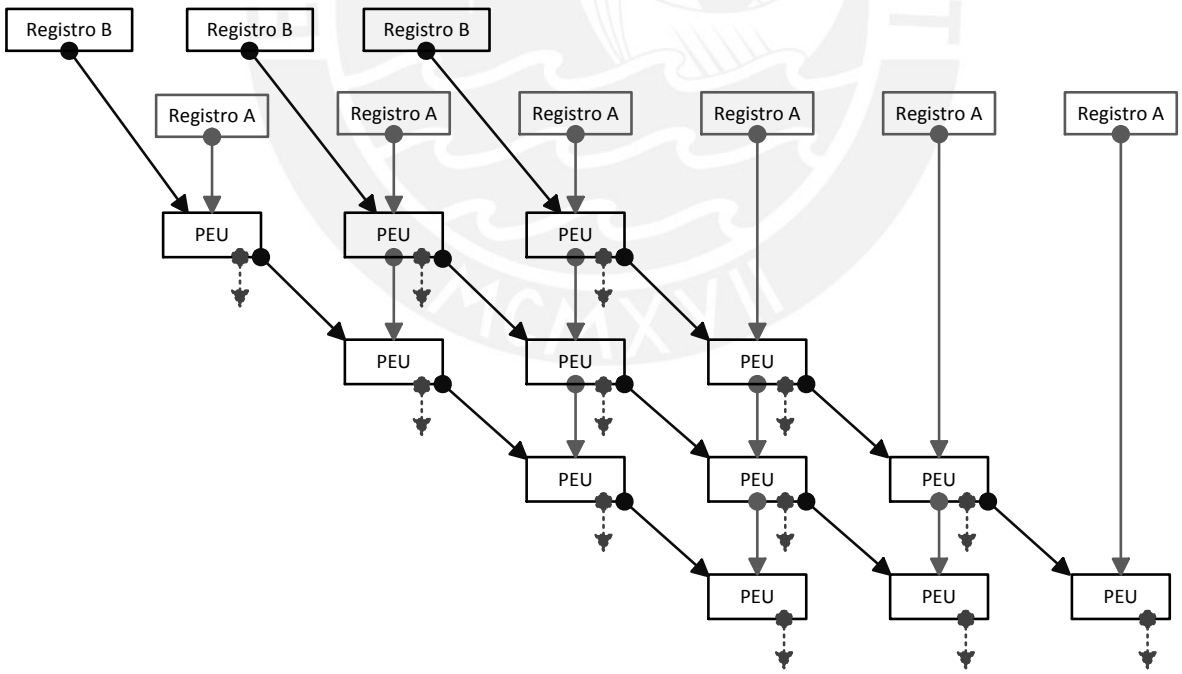


Figura 2.2: Esquema simplificado del arreglo sistólico

2.3. Procesador Elemental Unidimensional

De acuerdo a la sección anterior, un bloque PEU realiza la dilatación y erosión entre una fila de la imagen y una fila del elemento estructural. Sin embargo, se debe tener en cuenta que la fila de una imagen puede tener un gran número de píxeles, por lo cual, hacer la operación unidimensional para una fila de imagen con un solo bloque unidimensional puede tardar demasiado, debido a que, como se mostró en la sección, se debe recorrer cada píxel de la fila de la imagen. Debido a esto, cada bloque PEU está compuesto internamente por un número de PEs (Procesador Elemental). Cada PE realiza la dilatación o erosión de una fracción de la fila de imagen que se desea procesar.

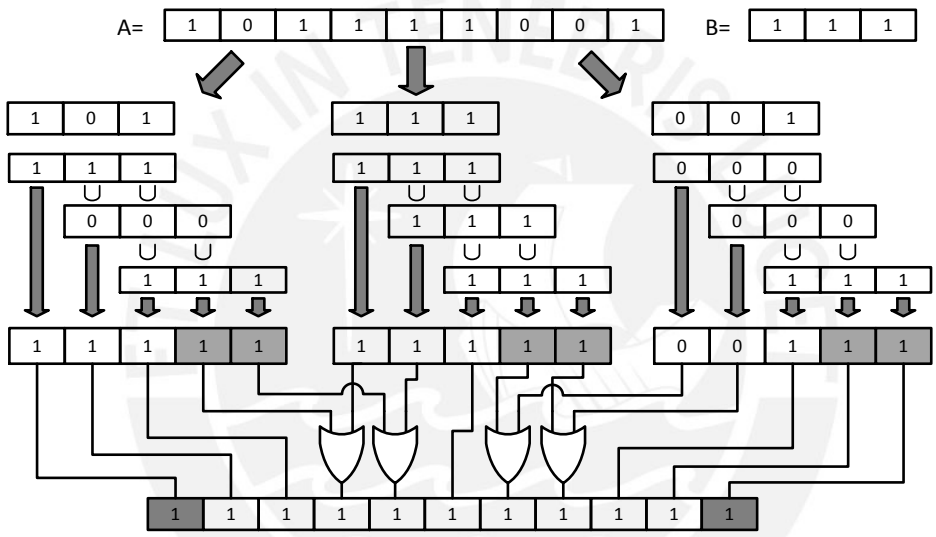


Figura 2.3: Ejemplo de concatenación para la dilatación

Los resultados parciales se pueden concatenar para hallar el resultado total. La concatenación consiste en unir o interceptar, de acuerdo a la operación, los píxeles que sobran en el extremo derecho con los primeros píxeles del extremo izquierdo del siguiente fragmento. La Figura 2.3 muestra este procedimiento para realizar la dilatación entre una fila de imagen de 9 píxeles agrupados en 3 fragmentos y un elemento estructural de 3 píxeles. Se observa que, de acuerdo a lo explicado anteriormente, el resultado de cada fragmento dilatado tiene un número de píxeles adicionales equivalente al tamaño del elemento estructural menos 1. Así, en este ejemplo hay dos píxeles adicionales en cada fragmento y son justo éstos píxeles los que se unen con los dos primeros píxeles del fragmento consecutivo. El resultado final que se muestra en la parte inferior presenta 11 píxeles, de los cuales se deben descartar los extremos que se encuentran sombreados, considerando al igual que en la Sección 1.4.1, que el origen del elemento estructural coincide con su centro geométrico.

Capítulo 3

Diseño del Sistema Propuesto

En este capítulo se explica el diseño de cada uno de los módulos que componen en coprocesador morfológico, desde la unidad básica PE hasta las interfaces de entrada y salida para la comunicación con el procesador.

3.1. Diseño del Procesador Elemental

Como fue explicado en la Sección 2.3, el PE es el circuito base de la unidad morfológica, ya que conforma los PEUs y éstos a su vez forman el arreglo sistólico. La Figura 3.1 muestra el módulo PE, indicando sus entradas y salidas. A y B son las entradas de los datos a operar, donde A corresponde a la imagen y B al elemento estructural. B es una entrada de X_SE_max bits, mientras que A es una entrada de sólo un bit, correspondiente al bit más significativo del registro de desplazamiento externo que contiene el fragmento de la fila de imagen que se desea operar. Por otro lado, *sel* y *opera_in* son entradas de control. *Sel* elige entre la operación de dilatación y erosión, mientras que *opera_in* habilita la operación del módulo PE cuando es "1". Las últimas dos entradas corresponden a la señal *reset* y a la señal de *clock*. En cuanto a las salidas, A_out y B_out contienen los mismos datos que ingresan por las entradas A y B y sirven para realizar las interconexiones entre los bloques PE que forman el arreglo sistólico. La salida C_out contiene el resultado de la operación realizada, dilatación o erosión. El tamaño de esta salida es de X_C_max bits. El parámetro X_SE_max, mencionado anteriormente, es una constante que permite al usuario indicar cuál es el tamaño horizontal máximo que tendrá el elemento estructural que será enviado por el procesador. X_C_max es otra constante definida como la suma de X_SE_max con *ancho_bus* menos 1, donde *ancho_bus* es una constante que indica el tamaño del fragmento de fila de imagen que se guarda en cada uno de los registros externos y que coincide con el ancho del bus de comunicación entre el procesador y el coprocesador, lo cual se explica con mayor detalle en la Sección 4.3.

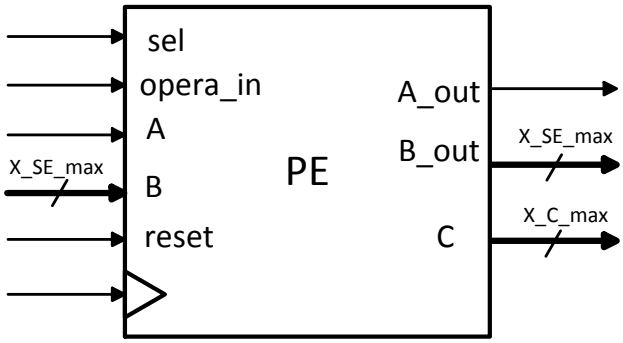


Figura 3.1: Módulo PE

La Figura 3.2 muestra el diseño interno del PE. Tanto a la entrada del bit correspondiente a la fila de la imagen como a la salida del registro interno, hay multiplexores. La función de éstos es seleccionar si tanto la entrada como la salida estarán negadas o si los datos pasarán de manera directa. Así, se implementan tanto la operación de dilatación como la de erosión, de acuerdo al concepto de dualidad presentado en la Sección 1.3.2. Se observa además, que dentro del PE hay un registro de desplazamiento, cuyo tamaño es definido por la constante *X_C_max*. Este registro funciona de manera sincronizada con el registro externo que contiene el fragmento de fila de imagen sobre el que se está trabajando. Ambos registros rotan un bit en sentido antihorario en cada ciclo de reloj. Así, en cada ciclo se tiene un nuevo píxel en la entrada *A* del módulo PE. Este píxel de entrada, luego de pasar por el primer multiplexor, usado para especificar qué operación realizar, sirve como selector de un segundo multiplexor. Así, de acuerdo al valor del píxel, se decide si se unirá el valor del elemento estructural al valor anterior guardado en el registro o si en su lugar, se unirán 0s, es decir, no se modificará el valor anterior del registro interno. De esta forma, se está realizando el procedimiento descrito en la Sección 1.4.1, con la única diferencia de que en este caso, en lugar de desplazar el elemento estructural por cada píxel de la imagen, es la imagen la que es desplazada.

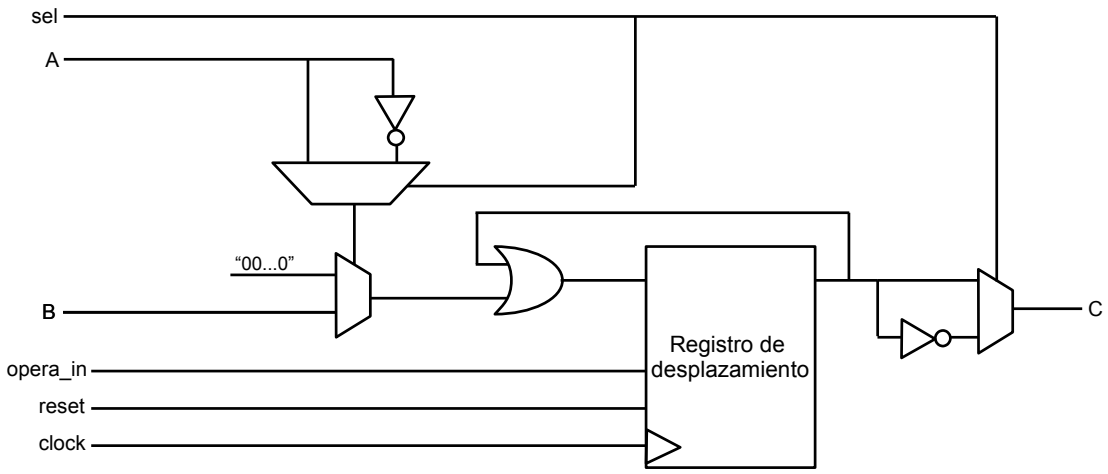


Figura 3.2: Estructura interna del PE

3.2. Diseño del Procesador Elemental Unidimensional

En la Sección 2.3 es mencionado que el PEU se encarga de realizar la dilatación entre una fila completa de una imagen y una fila del elemento estructural. Se indica también que este módulo está compuesto por varios PEs, de tal forma, que cada uno de éstos realiza la dilatación entre un fragmento de la fila de la imagen y la misma fila del elemento estructural. La Figura 3.3 muestra el módulo PEU, indicando sus entradas y salidas. Se observa que el número de entradas y salidas de este módulo es igual al existente en el módulo PE mostrado en la Figura 3.1. Además, los nombres de las señales son similares. Las señales *operación_in*, *sel*, *clock* y *reset* siguen siendo de un solo bit, mientras que las entradas y salidas correspondientes a los datos tienen un tamaño de bits equivalente a la multiplicación de su puerto correspondiente en el módulo PE por el número de PEs contenidos en el PEU. Los tamaños de las entradas y salidas también se muestran en la Figura 3.3. El parámetro *num_PEs* es una constante que indica el número de PEs que debe haber dentro de cada PEU.

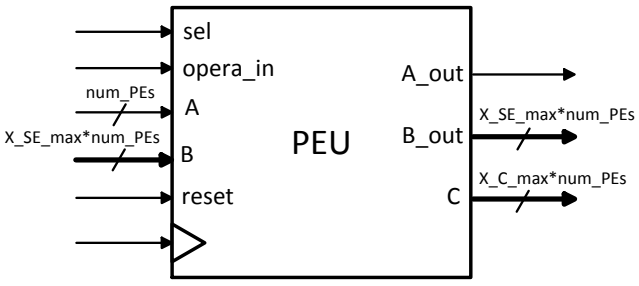


Figura 3.3: Módulo PEU

3.3. Módulo de Arreglo Sistólico

El módulo de arreglo sistólico consiste en el conjunto de PEUs distribuidos y unidos de acuerdo a la propuesta presentada en la Sección 2.2. Este módulo también incluye los registros de desplazamineto que almacenan los datos de las filas de la imagen de entrada, así como los registros de salida donde se almacenan los resultados parciales, lo cual será explicado en la Sección 3.4.

La Figura 3.4 muestra el módulo de arreglo sistólico, indicando sus entradas y salidas. La entrada *registro_A* corresponde a los fragmentos de diferente filas que son almacenados en los registros de desplazamiento internos. Así, el tamaño de cada fragmento de fila es igual al tamaño de los registros internos definido por el parámetro *ancho_bus*. Además, el número de entradas *registro_A* es igual al número de registros de desplazamiento internos, el cual está definido por: $(num\_PEUs\_x + num\_PEUs\_y - 1) * num\_PEs$. De igual forma, las entradas B corresponden a las filas del elemento estructural que se usarán en el arreglo. El número de entradas B es igual al número de PEUs que hay en cada fila del arreglo sistólico y que está definido por el parámetro *num_PEUs_x*. El tamaño de cada una de las entradas B está definido por el parámetro *X_SE_max*, el cual indica el número máximo de bits que tendrá cada fila del elemento estructural. Internamente, son sólo los PEUs de la primera fila del arreglo los que están conectados a los registros de entrada de imagen y a las entradas del elemento estructural, mientras que los PEUs de filas inferiores reciben estos datos a través de las salidas *A_out* y *B_out* de los PEUs ubicados en las fila superior inmediata, tal como es propuesto en la Figura 2.2.

En cuanto a las entradas de control del módulo de arreglo sistólico, se observa que aparte de las señales de *reset* y *clock*, hay cuatro más llamadas. Las señales *modo* y *operación_in* están conectadas a las entradas *sel* y *operación_in* de cada PEU, respectivamente. Así, con la entrada *modo* en baja se realiza la operación de dilatación mientras que cuando está en alta se ejecuta la erosión. Por otro lado, la entrada *operación_in* debe mantenerse en alta para que los PEs puedan operar, de lo contrario, éstos se encuentran deshabilitados. Por último, en el módulo de arreglo sistólico hay dos señales de dos bits cada una, *ctrl_A* y *ctrl_C*, las cuales sirven para indicar si los registros de entrada y salida deben mantenerse estáticos, cargar un nuevo dato o rotar.

3.4. Registros de Salida

Todos los módulos PEU que se encuentran en una misma fila del arreglo sistólico, contienen resultados que al unirse o intercectarse, darán como resultado la dilatación o erosión de una fila para la imagen procesada, respectivamente. Además, dentro de cada PEU, cada PE tiene resultados parciales que se deben concatenar. Por este motivo, por cada fila del

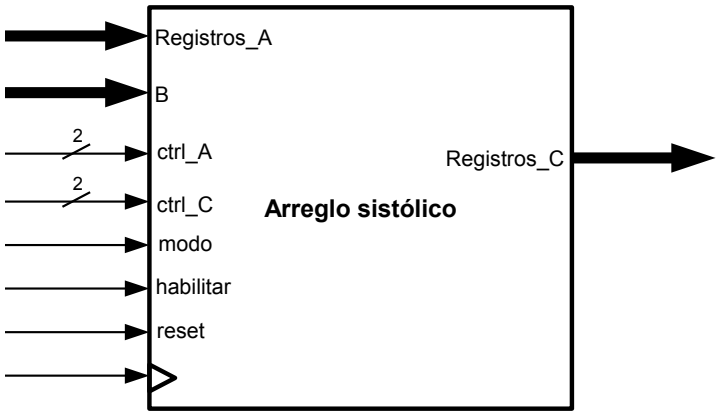


Figura 3.4: Módulo de arreglo sistólico

arreglo sistólico se tienen dos niveles de multiplexores y compuertas lógicas *AND* y *OR*, cuya salida final es almacenada en un registro.

Dentro de cada PEU, cada uno de los PEs tiene una ubicación, de tal forma que, los PEs ubicados en la misma posición dentro de diferentes PEUs, están operando con diferentes filas de la imagen, pero que pertenecen a las mismas columnas. Así, el primer nivel de multiplexor y compuertas *OR* y *AND*, se encarga de unir o intercectar el resultado de todos lo PEs ubicados en la misma posición dentro de distintos PEUs que forman una fila del arreglo sistólico. Luego de este nivel, se tendrán *num_PEs* grupos de datos, cada uno correspondiente a una misma fila de la imagen transformada, pero pertenecientes a diferentes columnas. Así, con el segundo nivel de multiplexor más compuertas *AND* y *OR* se concatenan a estos grupos, de acuerdo al procedimiento explicado en la Sección 2.3, consiguiendo así un único dato cuyo número de bits es igual a $(ancho\_mem \times num\_PEs + X\_SE\_\text{máx})$. Finalmente, este dato es unido o interceptado con los valores almacenados anteriormente en el registro y este nuevo resultado se almacena nuevamente en este registro. En la Sección 2.2 es mencionado que cuando el número de PEUs por fila del arreglo sistólico es menor que el número de filas del elemento estructural, se deben realizar operaciones consecutivas, cargando las diferentes filas de la imagen y del elemento estructural hasta realizar todas las combinaciones. Así, el registro incluido en cada fila del arreglo sistólico permite realizar esta iteración almacenando los resultados anteriores y uniéndolos o intercectándolos con los siguientes valores hasta cubrir todas las combinaciones. La Figura 3.5 muestra los dos niveles y el registro correspondiente a una fila de un arreglo sistólico donde el número de PEs es 4 y el número de PEUs es 3. Las salidas *sal_mux* del nivel 0 tienen un número de bits igual a la suma de *ancho_bus* con *X_SE_max* menos 1. Estas salidas son entradas del siguiente nivel. En el nivel 1, se realiza la concatenación de todos los fragmentos, dando como resultado las salidas *Sal_mux1*. Cada una de ellas tiene un número de bits igual a *ancho_bus*.

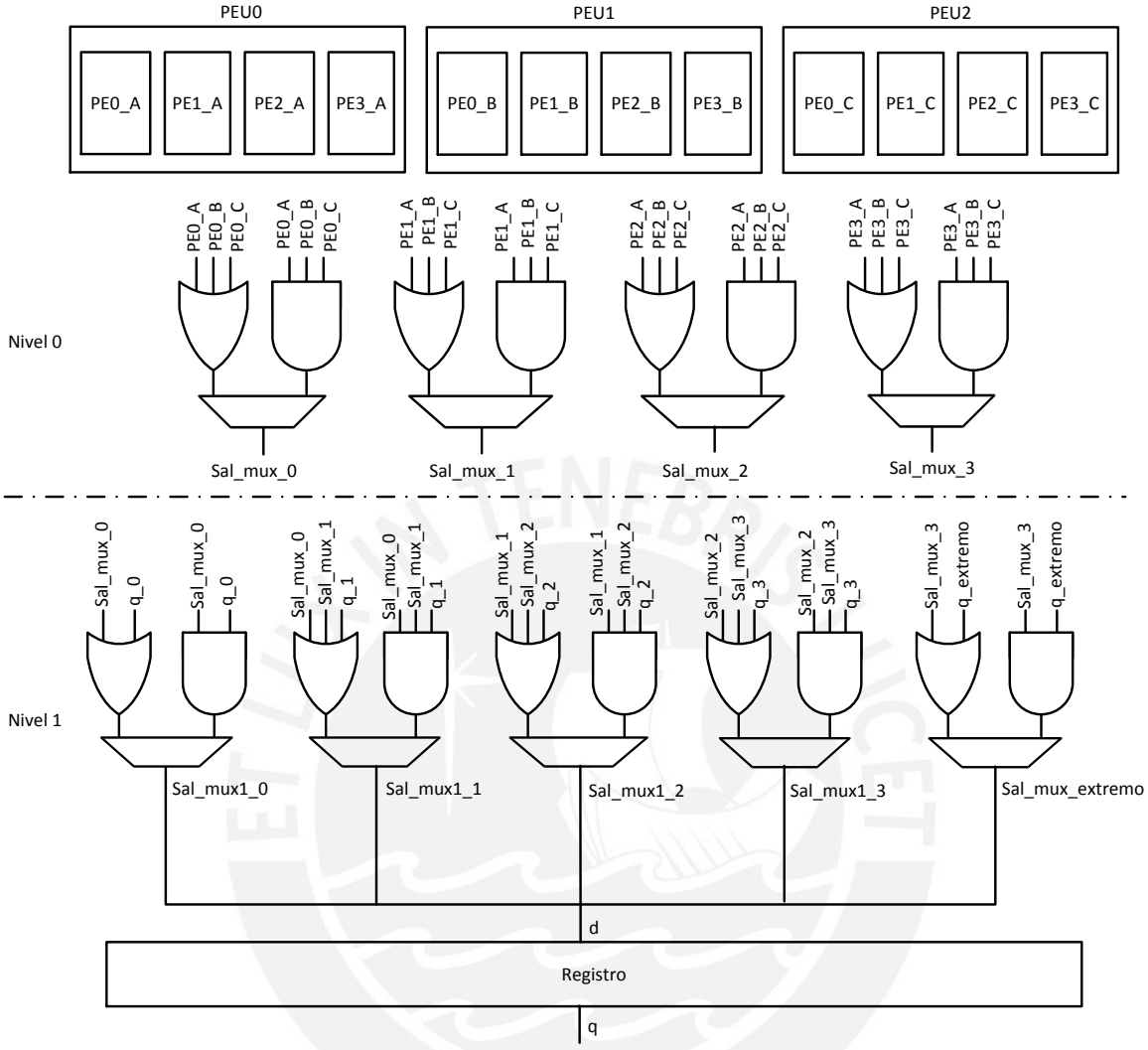


Figura 3.5: Etapa de salida de una fila del arreglo sistólico

3.5. Unidad de Control

De acuerdo a lo propuesto en la Sección 3.1, cada PE que forma el arreglo sistólico opera con un fragmento de imagen, cuyo tamaño está definido por el parámetro *ancho_bus*, que se encuentra almacenado en un registro de desplazamiento, desde donde es transferido a todos los PEs que se encuentran en una misma columna del arreglo sistólico. Sin embargo, se debe notar que cada PE sólo cuenta con los elementos necesarios para realizar la operación de erosión o dilatación con cada nuevo bit de la imagen que recibe cada ciclo, mas no es capaz de determinar si terminó la operación ni tampoco tiene control sobre el registro que almacena la imagen. Así, es necesario tener una unidad de control que controle los registros donde se almacena la imagen de entrada y también cada PE, de tal forma que todos estos

elementos puedan operar de manera sincronizada y sin errores. Además, esta unidad de control debe manipular los registros de salida de cada fila del arreglo sistólico, los cuales son explicados en la Sección 3.4.

La Figura 3.6 muestra la unidad de control diseñada, indicando sus entradas y salidas. La entrada *iniciar_operacion* indica el inicio de una nueva operación, mientras que *sel_operacion* indica si se debe realizar la operación de erosión o de dilatación. Las otras dos entradas corresponden a la señal de *reset* y *clock* comunes a todos los módulos. Las salidas de este módulo corresponden a las señales de control tanto de los módulos PE como de los registros de entrada de la imagen y de salida del resultado. Así, las salidas *sel* y *opera_in* están conectadas a las entradas del mismo nombre de cada PE. Por otro lado, la salida *ctrl_regA* está conectada a la entrada de control de cada uno de los registros A, que son los registros de desplazamiento que almacenan un fragmento de una fila de la imagen a operar. De la misma forma, la salida *ctrl_regC* está conectada a la entrada de control de cada uno de los registros C, que son los registros que se encuentran en cada fila del arreglo sistólico. Por último, la salida *fin* indica el fin de la operación.

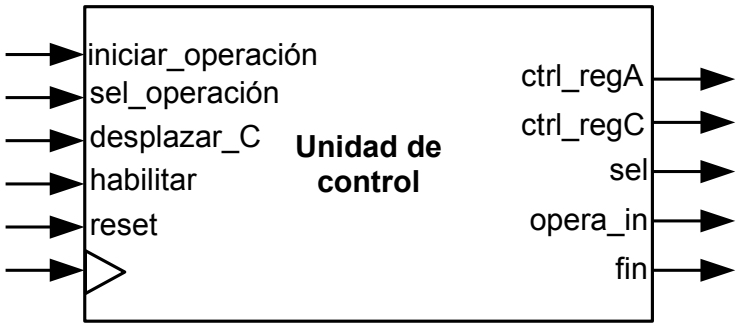


Figura 3.6: Unidad de control del arreglo sistólico

La Figura 3.7 muestra la máquina de estados que conforma la unidad de control con sólo cuatro estados. En el estado inicial todas las salidas están desactivadas, es decir, tanto los PEs como los registros están deshabilitados. Para pasar al estado *Cargar registros A*, la señal *iniciar_operacion* debe pasar a alta por lo menos por un ciclo de reloj. En el estado *Cargar registros A*, la señal *ctrl_regA* es colocada en alta, habilitando los registros para que carguen la imagen de entrada. En el siguiente pulso de reloj, se pasa al estado *Realizar operación*, en el cual se habilitan todos los PEs al colocar la señal *opera_in* en alta. Además, la salida *ctrl_regA* se coloca en "01", habilitando así el desplazamiento en sentido antihorario de los registros A. Para pasar al siguiente estado *Cargar registros C*, un contador externo deberá llegar al tope de su cuenta, donde el tope está definido por el parámetro *ancho_bus*, el cual establece el tamaño de los registros A, es decir, el tamaño de cada fragmento de fila. En el estado *Cargar registros C* se desactivan los PEs al colocar a baja la salida *opera_in*. Además, la salida *ctrl_regC* se coloca en "11", activando así los registros C para que carguen

los resultados obtenidos por cada PE. Finalmente, se regresa al estado de *Inicio*.

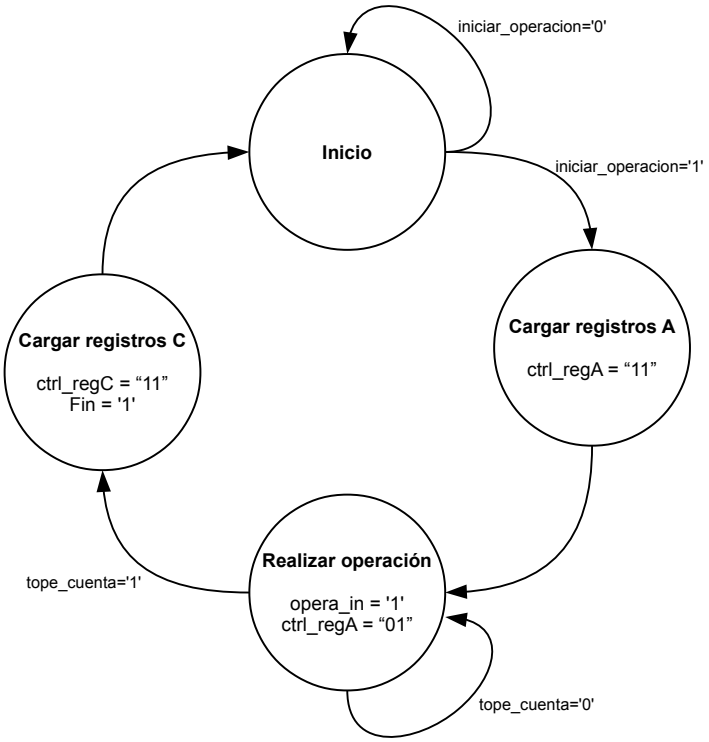


Figura 3.7: Máquina de estados de la unidad de control

3.6. Unidad Morfológica

La unidad morfológica está conformada por el módulo de control presentado en la Sección 3.5 y el módulo de arreglo sistólico explicado en la Sección 3.3. La Figura 3.8 muestra la interconexión de ambos módulos, formando así la unidad morfológica. Todas las señales de control del módulo de arreglo sistólico están conectadas a las salidas de control de la unidad de control. Por otro lado, todas las entradas del módulo de control forman parte de las entradas de la unidad morfológica. Así mismo, las entradas de datos del arreglo sistólico también forman parte de las entradas de la unidad morfológica. En cuanto a las salidas, se observa que son sólo dos. Una de datos correspondiente a los *ancho_bus* bits más significativos de cada registro de salida. La otra señal de salida de la unidad morfológica corresponde al módulo de control e indica que se ha terminado una operación. Esta unidad morfológica conforma el módulo más externo del coprocesador y es la que se debe comunicar con el procesador a través de una interfaz.

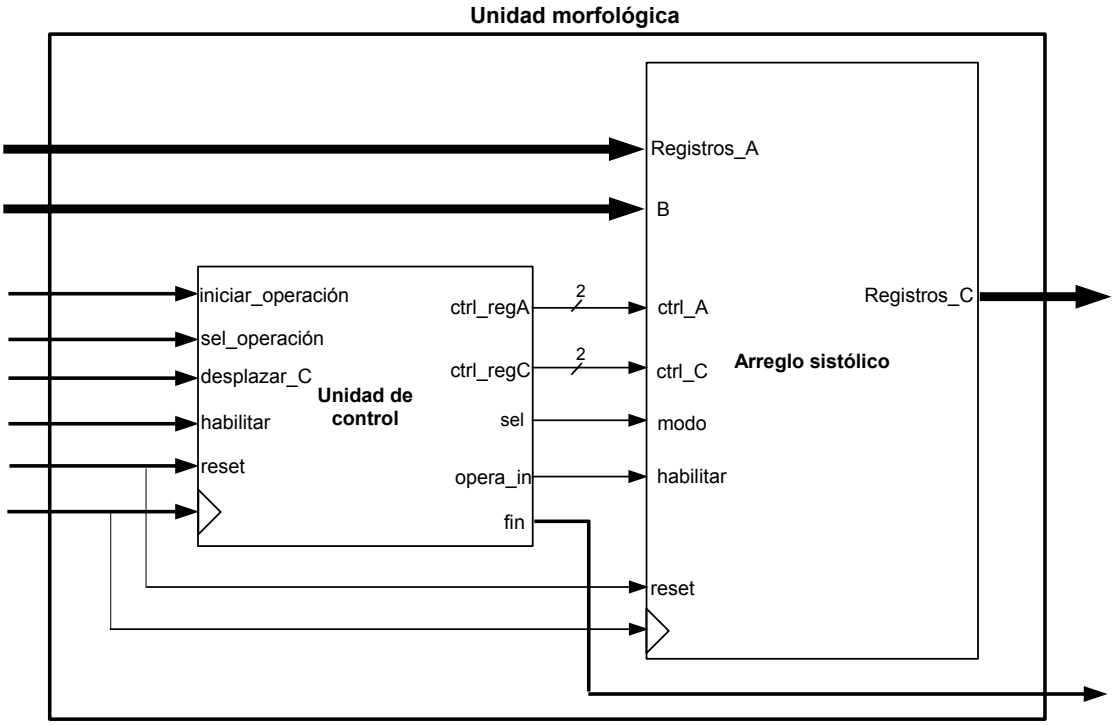


Figura 3.8: Diseño interno de la unidad morfológica

Capítulo 4

Resultados y Análisis

El diseño de un circuito en VHDL consiste de cuatro etapas: síntesis, traducción, mapeo y ruteo. En este capítulo se muestran los resultados conseguidos en la primera etapa de síntesis, obtenidos para el coprocesador usando diferentes parámetros. Además, se presentan las simulaciones correspondientes a la verificación funcional, conocida como *testbench*.

4.1. Síntesis del circuito

4.1.1. Visor RTL

En esta sección se muestran los resultados obtenidos al sintetizar la descripción de hardware realizada en VHDL. Mediante la herramienta RTL viewer, disponible tanto en el entorno de desarrollo ISE de Xilinx como Quartus II de Altera, se puede verificar mediante diagramas de módulos y sus interconexiones, que el circuito creado por el compilador cumple con las especificaciones del circuito propuesto, es decir, permite verificar que la descripción realizada en VHDL es correcta. Además, al sintetizar para diferentes valores de los parámetros descritos en secciones anteriores, se puede verificar que el circuito cumple con ser portable y parametrizable.

A continuación, se muestran algunos de los resultados obtenidos para distintos parámetros. La Figura 4.1 muestra el módulo correspondiente a un PE sintetizado para *ancho_bus*=32 y *X_SE_max*=3. Se observa que el módulo generado por el *RTL Viewer* del software *Quartus II* tiene las mismas entradas y salidas que el módulo propuesto en la Figura 3.1. Además, la arquitectura interna consta de tres multiplexores, compuertas *OR* y *NOT* y un registro interno de *X_C_max* bits, como muestra la Figura 3.2.

Luego de haber analizado los módulos PE, se verifican los módulos PEU, también obtenidos mediante la herramienta *RTL Viewer* del software *Quartus II*. Como es explicado en la Sección 3.2, cada módulo PEU consiste en un grupo de módulos PE, cada uno traba-

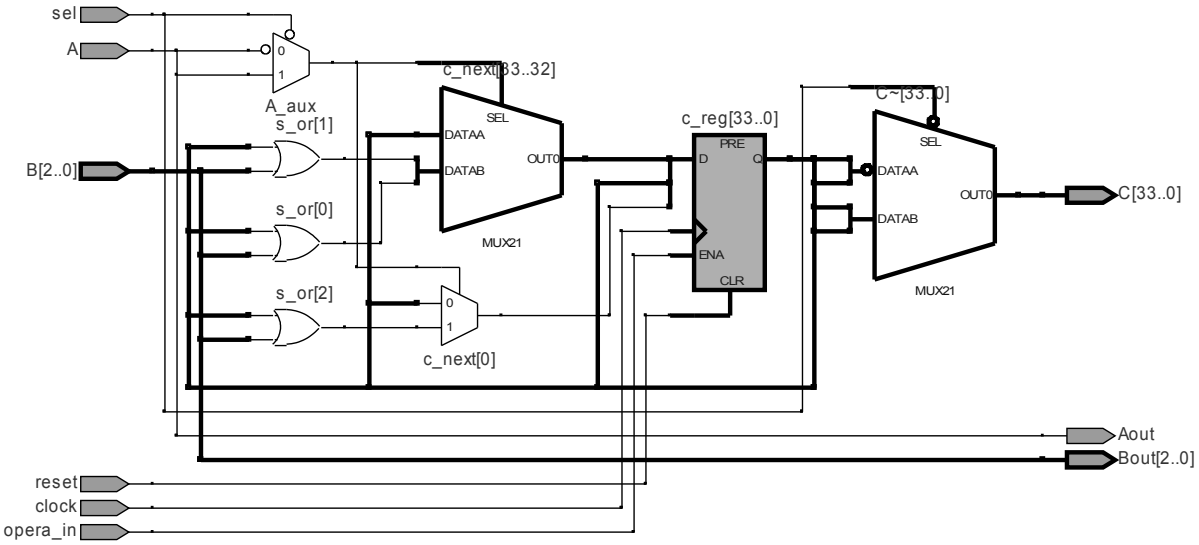


Figura 4.1: Módulo PE sintetizado para *ancho_bus*=32 y *X_SE_max*=3

jando con un fragmento de fila diferente, pero con una misma fila del elemento estructural. El número de PEs dentro de un PEU es definido por la constante *num_PEs*. La Figura 4.2 muestra la estructura interna obtenida para *num_PEs*=2. Se observa que las entradas y salidas están de acuerdo al módulo PEU propuesto en la Figura 3.3.

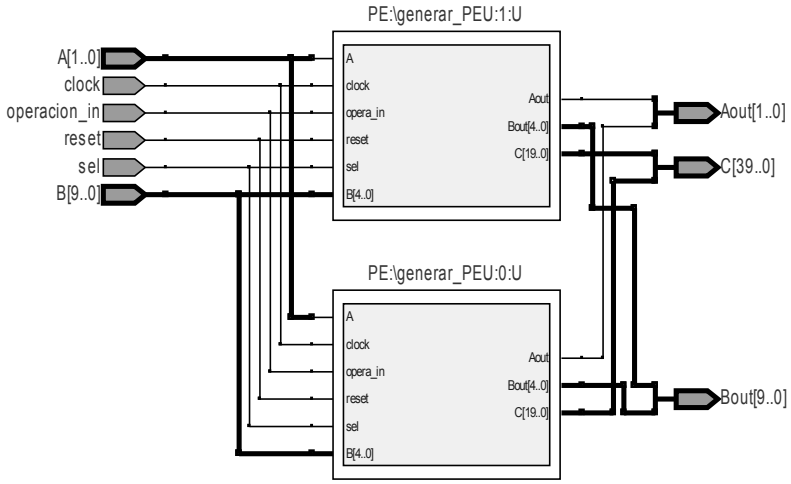


Figura 4.2: Módulo PEU sintetizado para *num_PEs*=2

Finalmente, se muestran los resultados obtenidos para el módulo del arreglo sístólico. En su interior se muestra un número de PEUs que es determinado por los parámetros *num_PEUs_x* y *num_PEUs_y*. La Figura 4.3 muestra el arreglo sistólico sintetizado para para *num_PEUs_x*=3 y *num_PEUs_y*=4. Se observa entonces que el número de PEUs dentro del arreglo sistólico es equivalente a la multiplicación de *num_PEUs_x* con *num_PEUs_y*, dando así para este caso un número total de 12 PEUs. Es posible verificar también que el número

de salidas del arreglo sistólico es igual al parámetro *num_PEUs_y*. Estas salidas son las que pasan por los dos niveles de multiplexores y compuertas *AND* y *OR*, y posteriormente a un registro, como es explicado en la Sección 3.4.

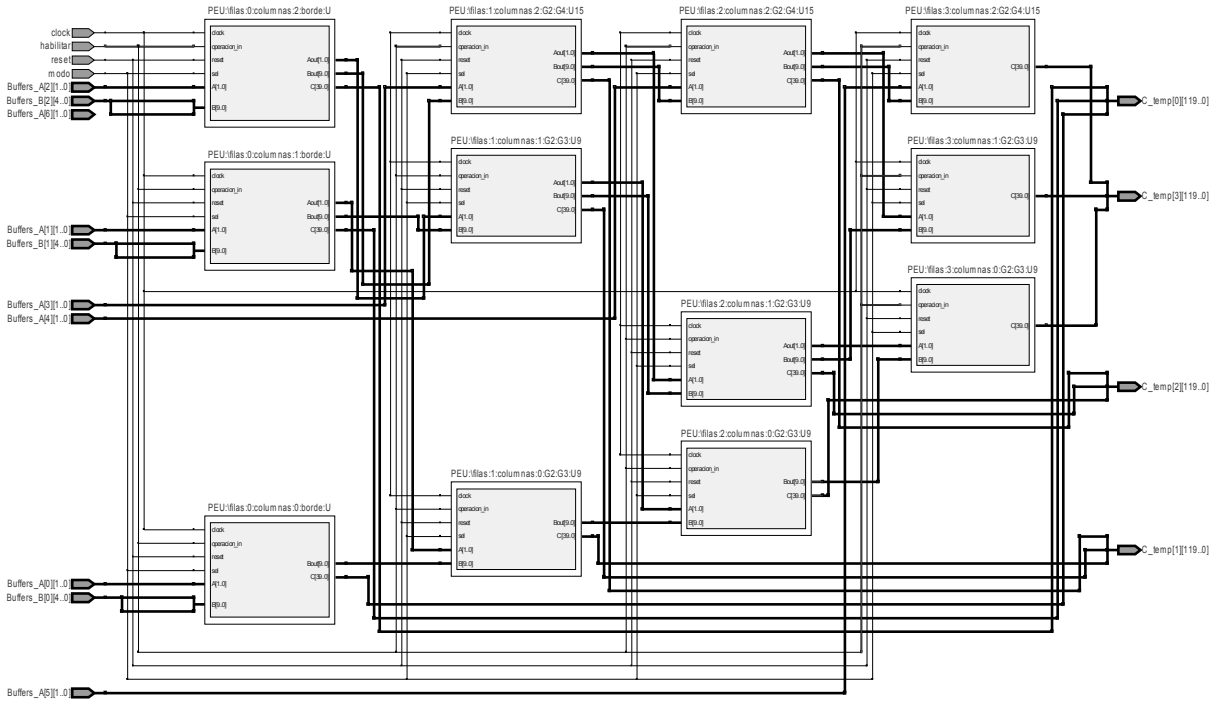


Figura 4.3: Arreglo sistólico para *num_PEUs_x*=3 y *num_PEUs_y*=4

4.1.2. Reporte de Síntesis

Otro documento que se genera al realizar la síntesis de la descripción realizada en VHDL, tanto en el entorno *Quartus II* como el *ISE*, es el reporte de síntesis. Este archivo contiene datos importantes relacionados con el uso de recursos lógicos, de acuerdo a las características del FPGA para el cual se realizó la síntesis. Por otro lado, este reporte también brinda información sobre los retardos existentes en el circuito y es capaz de calcular cuál es la máxima frecuencia de operación posible para el circuito sintetizado. En esta sección, se incluyen los datos obtenidos al sintetizar el circuito para diferentes parámetros y distintas plataformas.

La Tabla 4.1 muestra los resultados obtenidos al sintetizar el circuito para un FPGA Spartan 6, incluido en la tarjeta de desarrollo *Atlys* de *Xilinx*. Para los primeros cuatro resultados, sólo se varió el parámetro *num_PEUs_y*, mientras que para los últimos resultados sólo se cambió el parámetro *num_PEUs_x*. Al aumentar alguno de estos parámetros, aumenta el número de PEUs que conforman el arreglo sistólico. Por este motivo, al aumentar alguno de

estos parámetros, tanto el número de Look Up Tables (LUTs) como el número de registros, aumentan en aproximadamente la misma proporción (ver Tabla 4.1). Por otro lado, se observa que al aumentar estos parámetros, la máxima frecuencia de operación disminuye. Esto se debe a que al aumentar el número de PEUs, el número de interconexiones también aumenta, dando como resultado un incremento de los retardos. Así, la máxima frecuencia de operación determinada para que el circuito opere sin errores, disminuye.

Tabla 4.1: Consumo de recursos del coprocesador en un FPGA Spartan 6

num_PEs	num_PEUs_x	num_PEUs_y	Slice registers	Slice LUTs	Max. frec. (MHz)
4	3	4	1270 (2 %)	1313 (4 %)	561.230
4	3	8	2386 (4 %)	2497 (9 %)	513.690
4	3	16	4618 (8 %)	4865 (17 %)	437.103
4	5	4	1954 (3 %)	2009 (7 %)	535.432
4	9	4	3338 (6 %)	3529 (12 %)	472.312
4	15	4	5414 (9 %)	5753 (21 %)	421.131

De acuerdo al concepto de portabilidad, el circuito elaborado debe poder ser sintetizado para cualquier plataforma. Como forma de verificación, el circuito fue sintetizado también para el FPGA *Cyclone II*, incluido en la tarjeta de desarrollo *DE2* de la marca *Altera*. Los resultados obtenidos se muestran en la Tabla 4.2. Así, se han cubierto las dos principales marcas desarrolladoras de FPGAs, Xilinx y Altera. La Tabla 4.2 muestra el consumo de elementos lógicos para el FPGA *Cyclone II*, usando los mismos parámetros empleados para los resultados de la Tabla 4.1. Finalmente, en la tabla 4.3 se comparan los recursos usados por el sistema propuesto en este trabajo y la arquitectura desarrollada en [15].

Tabla 4.2: Consumo de recursos del coprocesador en un FPGA Cyclone II

num_PEs	num_PEUs_x	num_PEUs_y	Elementos lógicos
4	3	4	1308 (4 %)
4	3	8	2456 (7 %)
4	3	16	4752 (14 %)
4	5	4	2068 (6 %)
4	9	4	3610 (11 %)
4	15	4	5865 (18 %)

Tabla 4.3: Comparación de recursos en tarjeta Xilinx Virtex XCV1000

X_SE_max	LUTs del sistema propuesto	LUTs del sistema desarrollado en [15] para dilatación	LUTs del sistema desarrollado en [15] para erosión
5	681	129	132
11	825	325	339
21	1113	898	897
31	1313	1829	1813

4.2. Verificación Funcional

El proceso de verificación funcional consiste en colocar estímulos al sistema a probar y comprobar si los resultados obtenidos son correctos. Para poder llevar a cabo este proceso se utilizan las herramientas de software proporcionadas por los lenguajes de descripción de hardware. Este proceso de verificación, conocido como testbench, permite ingresar estímulos al circuito y comparar la respuesta obtenida con valores de referencia, indicando de manera automática la existencia de alguna falla. En este caso, se han generado vectores dentro del mismo testbench para las señales de control, mientras que para los datos de la imagen y los vectores de referencia se han usado archivos de texto externos. Para la lectura de estos archivos de texto se ha usado una librería estándar de VHDL llamada *textio*, la cual permite acceder al contenido de un archivo de texto por filas. La Figura 4.4 muestra la metodología seguida para realizar estas pruebas. Se observa que afuera del testbench se tiene un archivo de texto con los vectores de prueba y los resultados esperados. Dentro del testbench hay tres módulos: a) generación de formas de onda, b) modelo bajo prueba y c) comparación de resultados. El primero de ellos, generación de formas de onda, se encarga de leer los vectores de prueba del archivo de texto y enviarlos a las entradas de datos de imagen del módulo denominado modelo bajo prueba, que en este caso corresponde al coprocesador. Además de leer y transferir los datos guardados en el archivo de texto, el módulo de generación de formas de onda también se encarga de generar las señales de control necesarias para que opere el coprocesador. Finalmente, las respuestas obtenidas a la salida del módulo bajo prueba son enviadas al módulo de comparación, el cual se encarga de verificar que el resultado obtenido es igual al valor esperado, que fue leído previamente por el módulo de generación de formas de onda. Finalmente, de acuerdo al resultado de la verificación, se envía una indicación de aprobación o falla.

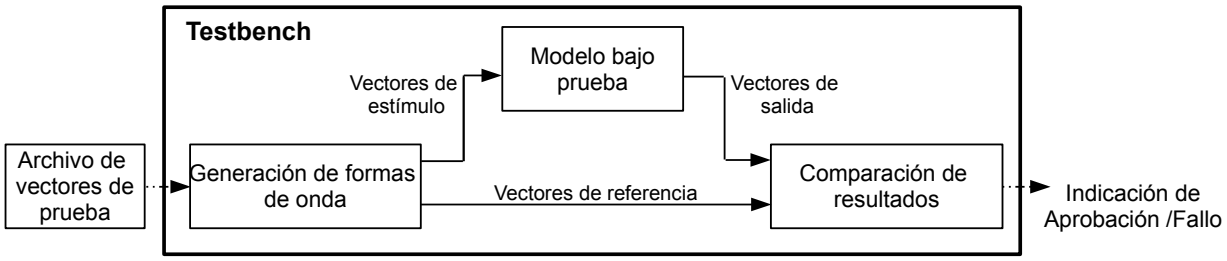


Figura 4.4: Metodología usada para el *testbench*

4.2.1. Generación de Vectores de Prueba

En este trabajo los vectores de prueba corresponden a filas o secciones de imágenes binarias. Así, para poder realizar el *testbench*, fue elaborado un banco de imágenes binarias y almacenadas en un archivo de texto. Para crear este banco de imágenes se ha usado el software Matlab. El procedimiento comienza a partir de una imagen en escala de grises. El primer paso consiste en añadir ruido a la imagen que se encuentra en escala de grises usando una función de Matlab llamada *imnoise*, agregando como parámetro el tipo de ruido que se desea, por ejemplo, *salt&pepper*, correspondiente al ruido *sal y pimienta*. Después de esta operación, se tiene la imagen con ruido en escala de grises. A continuación, esta imagen es convertida a binario mediante un valor umbral, usando los comandos de comparación de Matlab. Luego de esto, se tiene una matriz con elementos del tipo lógico, correspondiente a una imagen binaria. Por último, esta imagen, junto con su erosión o dilatación obtenida con el comando *imerode* o *imdilate*, se graba en un archivo de texto usando el comando *dlmwrite*. Así, al final de este procedimiento se tiene en un archivo de texto la imagen binaria y su correspondiente dilatación o erosión. La Figura 4.5 muestra el procedimiento antes descrito, mostrando las variaciones que sufre la imagen en cada etapa.

4.2.2. Simulación Funcional

De acuerdo a lo propuesto en la Sección 3.5, la unidad morfológica está conformada por una unidad de control y un número de PEs, agrupados dentro de PEUs. Además, la unidad morfológica contiene dos tipos de registros, unos para almacenar un fragmento de la imagen de entrada y otros para almacenar los resultados parciales obtenidos a la salida de los PEs. Se vió además, que la unidad de control interactúa con cada uno de los PEs y registros, de manera simultánea. La Figura 4.6 muestra el diagrama de simulación obtenido para la unidad de control interactuando con un PE. Las seis primeras señales mostradas en el diagrama corresponden a las entradas de la unidad de control, de acuerdo al módulo mostrado en la Figura 3.6. Por otro lado, las entradas A y B corresponden a una fracción de

Figura 4.5: Ejemplo del procedimiento para generar archivo de texto para pruebas

fila de imagen de entrada y a una fila del elemento estructural, respectivamente. Estos son los datos de entrada al PE. Finalmente, las señales de salida *C_PE_out*, *C* y *fin* corresponden a la salida del PE, al registro de salida y a la indicación de fin de la operación, respectivamente. En esta simulación, el fragmento de entrada de imagen es de 8 bits, mientras que la fila del elemento estructural es de 5 bits. Se observa que la simulación inicia con la entrada *reset* en alta, la cual coloca en 0 los registros de entrada y salida de la imagen, así como el contador interno de la unidad de control. En el siguiente periodo, las entradas *habilitar* e *iniciar_operación* se colocan en alta. La señal *habilitar* debe permanecer en alta durante todo el proceso, mientras que la señal *iniciar_operación* sólo necesita estar en alta durante un ciclo de reloj. En el siguiente periodo, luego de activar *iniciar_operación*, la máquina de estados de la unidad de control pasa inicia su operación, pasando al estado de carga del registro de la imagen de entrada. Esto sólo dura un ciclo, así que en el siguiente periodo de reloj, se pasa al estado de operación, activando el PE y el contador. Luego de 8 ciclos de reloj, el contador del módulo de control llega a su tope, pasando así al siguiente estado, que corresponde a la carga del resultado final *C_PE_out* en el registro *C*.

En general, el número de ciclos que transcurren desde que la señal *iniciar_operación* se coloca en alta es quivalente al número de bits que conforman un fragmento de imagen, indicado por el parámetro *ancho_bus*, más 2, donde la adición de 2 corresponde al periodo de carga del fragmento de imagen en el registro de entrada más el periodo de carga del resultado del PE en el registro de salida. De la misma forma como es mostrado en la simulación para un PE es que funcionan todos los PEs en simultáneo, pasando sus resultados por los dos niveles de multiplexores y compuertas *OR* y *AND* indicados en la Sección 3.4 y finalmente grabándose en el registro de salida correspondiente a su fila. De acuerdo a los parámetros *num_PEUs_x*, *num_PEUs_y* y *num_PEs* queda determinado el número de PEs totales que

debe tener el arreglo sistólico. A mayor número de PEs, mayor será la cantidad de datos procesados simultáneamente y menor el número de iteraciones necesarias, haciendo así el procesamineto más rápido.

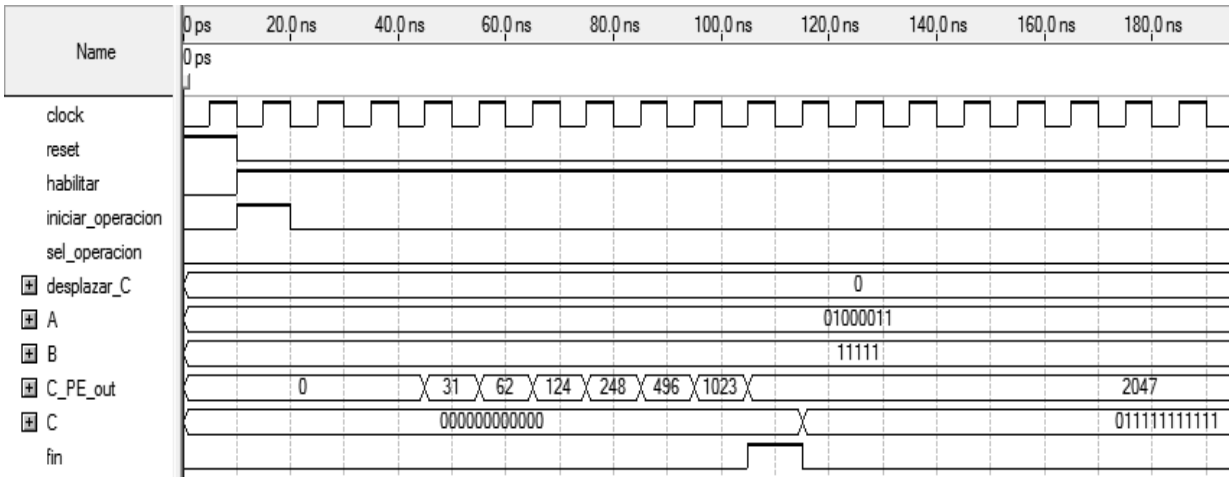


Figura 4.6: Funcionamiento de un módulo PE con la unidad de control

La Ecuación 4.1 muestra cómo es calculado el número de píxeles procesados simultáneamente en una operación de la unidad morfológica. Por otro lado, la Ecuación 4.2 muestra el cálculo del número de ciclos que tarda en enviarse todos los datos al coprocesador. De la misma forma, la Ecuación 4.3 muestra el cálculo de ciclos que tarda en enviarse los datos hacia el procesador. Finalmente, la Ecuación 4.4 muestra el número de ciclos que demora la unidad morfológica en terminar una operación. En base a estas ecuaciones y por medio de la verificación funcional, se obtiene la rapidez de procesamiento para diferentes valores de los parámetros *num_PEUs_x*, *num_PEUs_y* y *num_PEs*, así como para diferentes valores del parámetro *Y_SE* que indica el número de filas del elemento estructural. Para los resultados mostrados en la Tabla 4.4, el *ancho_bus* es igual a 32 y la frecuencia del reloj es de 100 MHz.

$$num\_PEUs\_y \bullet num\_PEs \bullet ancho\_bus \tag{4.1}$$

$$\frac{Y\_SE}{num\_PEUs\_x}\left[num\_PEs \bullet (num\_PEUs\_x + num\_PEUs\_y - 1)\right] \tag{4.2}$$

$$num\_PEUs\_y \bullet num\_PEs \tag{4.3}$$

$$\frac{Y\_SE}{num\_PEUs\_x}\left[ancho\_bus + 2\right] \tag{4.4}$$

Tabla 4.4: Rapidez de procesamiento

num_PEs	num_PEUs_x	num_PEUs_y	Y_SE	Rapidez (Gbps)
4	3	4	3	0.691
4	5	4	5	0.624
4	3	4	9	0.360
4	5	4	15	0.341
8	3	4	3	0.898
16	5	4	5	0.906
4	3	8	3	0.966
4	5	16	5	1.15

4.3. Verificación Experimental

Dado que para este trabajo no se cuenta con un procesador que pueda interactuar directamente con el coprocesador diseñado, para efectos de verificar el funcionamiento del coprocesador propuesto es necesario adicionar una interfaz de comunicación, de acuerdo a las características del procesador *Microblaze* y del bus *Fast Simplex Link* disponibles en el FPGA Spartan 6 de la tarjeta de desarrollo *Atlys*. *Microblaze* es un procesador de 32 bits de arquitectura *Hardvard*. Tiene 32 registros internos de 32 bits cada uno, una unidad de control y una unidad aritmética lógica. Adicionalmente, es posible incluir un multiplicador y divisor al sintetizar el circuito. Las instrucciones que serán ejecutadas por el procesador son almacenadas en una sección del FPGA conocida como *Block RAM*.

La Figura 4.7 muestra la estrategia usada para realizar las pruebas. Se observa que la conexión entre el coprocesador y el procesador se da a través de la interfaz de comunicación. Tanto el procesador *Microblaze*, como el coprocesador y la interfaz de comunicación se encuentran dentro del FPGA. Fuera del FPGA se encuentra la memoria DDR2 donde se almacena la imagen que se desea procesar y el resultado obtenido.

La Figura 4.8 muestra la interfaz *Fast Simplex Link* y sus señales de control. Esta interfaz se basa en una arquitectura FIFO y se ha diseñado exclusivamente para aplicaciones de aceleración por hardware, permitiendo una operación de hasta 600 MHz. Se observa en el lado izquierdo las señales correspondientes al maestro, mientras que en la parte derecha se muestran las señales del esclavo. La entrada *FSL_M_Data* corresponde a la entrada de datos de 32 bits, mientras que la salida *FSL_S_Data* corresponde a la salida de datos de 32 bits. Las señales *FSL_M_write* y *FSL_S_Read* sirven para realizar la escritura o lectura de algún dato en el bus, respectivamente. Otras señales importantes son *FSL_M_control* y *FSL_S_control*, las cuales sirven para indicar si los datos recibidos o enviados corresponden

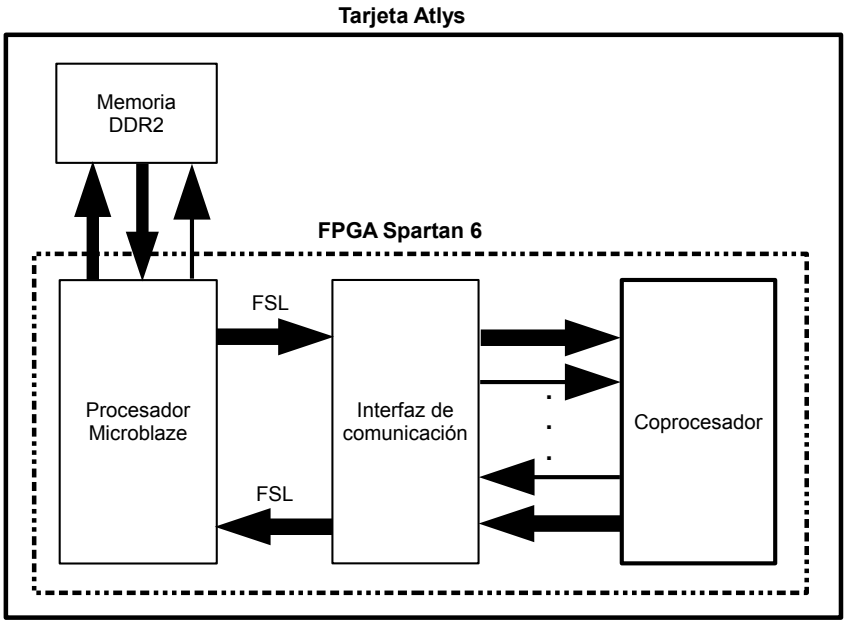


Figura 4.7: Esquema usado para la verificación experimental

a instrucciones o realmente a datos. Por otro lado, las señales *FSL_M_Full* y *FSL_S_Exists* indican cuando el bus se encuentra lleno o vacío, respectivamente. Por último, se muestran las señales *FSL_M_Clk*, *FSL_S_Clk* y *FSL_Clk*, correspondientes a las entradas de reloj. En este caso, el procesador y el coprocesador trabajan a la misma frecuencia de 100 MHz, por lo cual sólo se usa la señal *FSL_Clk*, empleada cuando hay sincronismo entre ambas entidades. Así, las señales *FSL_M_Clk* y *FSL_S_Clk* se dejan libres, ya que éstas sólo se emplean cuando ambos módulos operan de manera asíncrona.

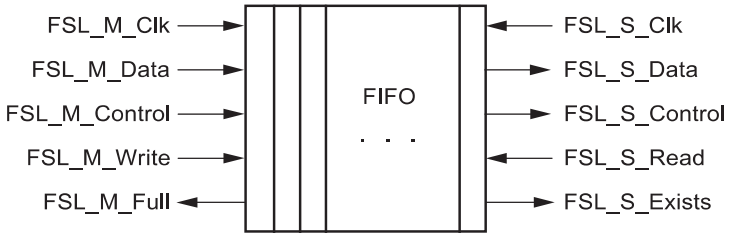


Figura 4.8: Señales de la interfaz Fast Simplex Link

Las instrucciones, mostradas en la Tabla 2.1, son enviadas desde el procesador, el cual además se encarga de colocar en alta la señal *FSL_M_control*, lo cual permite diferenciar entre el envío de instrucciones y el de fragmentos de la imagen. Una vez iniciada la operación, el procesador *Microblaze* se encarga de leer fragmentos de la imagen que está almacenda en la memoria DDR2 y enviarlos hacia el coprocesador. Así mismo, el procesador recibe los

fragmentos procesados de la imagen y los almacena en otra sección de la memoria DDR2. La Figura 4.9 muestra la máquina de estados correspondiente a la transferencia de datos desde el procesador hacia el coprocesador y viceversa. La operación inicia cuando la señal operación, que es manipulada por el procesador, pasa a alta. El siguiente estado, lectura de datos, corresponde a la lectura de datos de imagen enviados por el procesador. En total, son *num_PEUs_x* por *num_PEs* fragmentos de la de imagen recibidos de manera consecutiva. Así, cuando el contador *nr_of_reads* llega a 0 indica que se leyeron la cantidad de datos consecutivos de imagen necesarios, los cuales deben haber sido enviados por el procesador. Entonces, una vez que se alcanza el 0, ya se encuentran todos los datos almacenados en los registros de entrada y se pasa al estado *Activar unidad morfológica*. En este estado, se activa la unidad morfológica explicada en la Sección 3.6, colocando en alta su señal de entrada iniciar operación. Una vez activada la unidad morfológica, se pasa al estado Espera, en el cual se permanece hasta que la señal *fin* de la unidad morfológica cambie a alta, indicando así el fin de la operación. Por último, en el estado Escritura de resultados se envían al procesador los resultados almacenados en los registros de salida de la unidad morfológica.

Finalmente, la Figura 4.10 muestra el resultado obtenido para cada una de las seis operaciones morfológicas realizadas por el coprocesador para una misma imagen de 480 x 480 píxeles, usando un elemento estructural de 3 x 3.

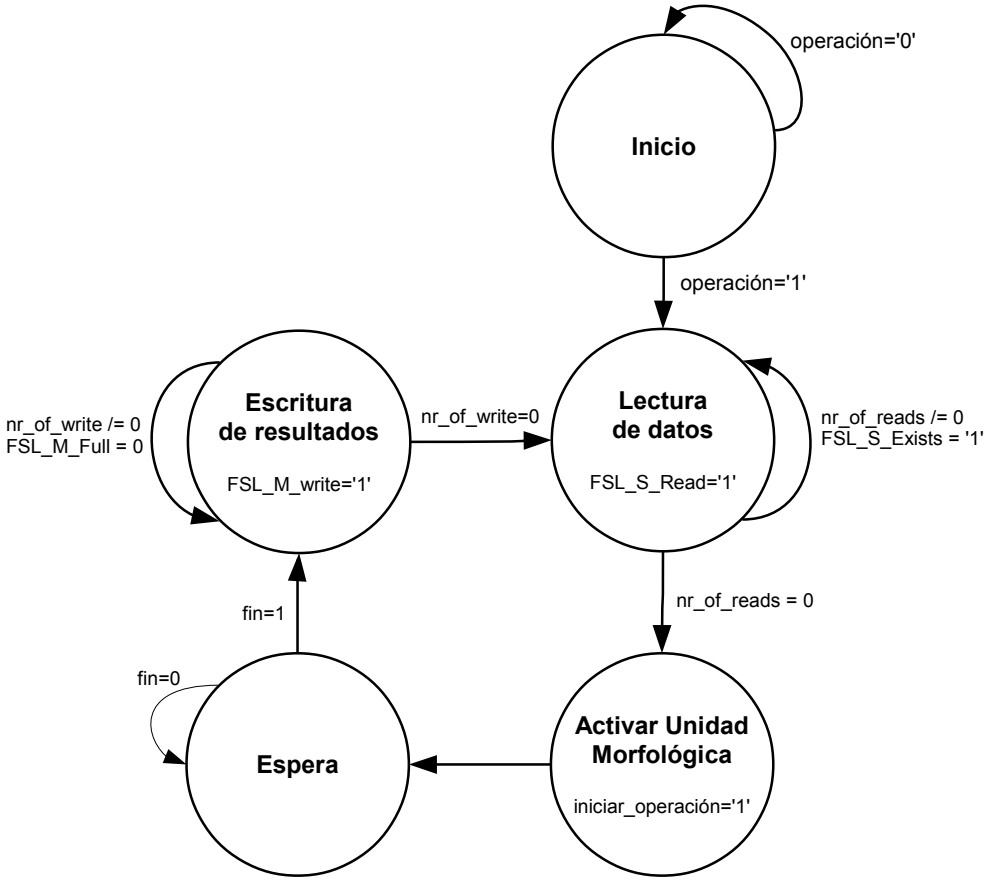
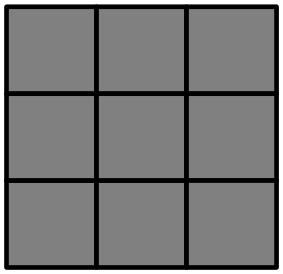


Figura 4.9: Máquina de estados para la transferencia de datos entre el procesador y el coprocesador

(a) Imagen original



(b) Elemento estructural

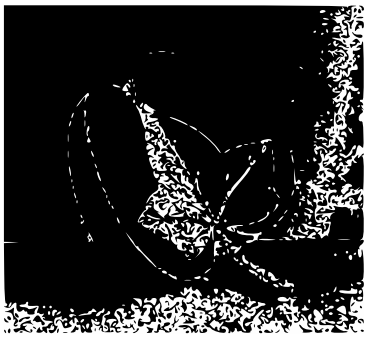
(c) Erosión

(d) Dilatación

(e) Apertura

(f) Cerradura

(g) Gradiente interno



(h) Gradiente externo

Figura 4.10: Resultados de las operaciones morfológicas

Conclusiones

Mediante los conceptos de operadores morfológicos unidimensionales y bidimensionales, se ha conseguido separar procesos de tal forma que ha sido posible formular una arquitectura basada en arreglos sistólicos que permite desarrollar las operaciones básicas de dilatación y erosión. El arreglo sistólico bidimensional elaborado no sólo permite la ejecución de operaciones en simultáneo, sino que su forma y flujo de datos simétrico ha permitido el desarrollo de una descripción de *hardware* totalmente parametrizable.

Se ha verificado la portabilidad de la descripción realizada, sintetizándola y simulándola para diversas plataformas tanto de la marca Altera como de Xilinx. Además, dentro del concepto de portabilidad, se ha cumplido también con realizar un sistema totalmente parametrizable. En este sentido, se ha verificado que la descripción permite, mediante la variación de parámetros, modificar el número de PEUs tanto por filas como por columnas, así como el número de PEs dentro de cada PEU.

El consumo de elementos lógicos del PE diseñado es bastante bajo, ya que éste sólo cuenta con un grupo de compuertas lógicas *AND* y *NOT*, 3 multiplexores y un solo registro interno donde se almacena el resultado parcial. Los registros que almacenan la imagen de entrada y los resultados finales son externos a los PEs y compartidos por todos los que se encuentran en una misma fila o columna del arreglo sistólico. Esta arquitectura junto con el desarrollo de una sola unidad de control simple y externa a los PEs ha permitido que la unidad morfológica tenga un bajo consumo de recursos. Así, en las Tablas 4.1 y 4.2 es mostrado el bajo consumo de recursos de esta unidad sintetizada en plataformas que poseen un nivel medio de recursos, logrando así sintetizar un arreglo sistólico con hasta 3840 PEs.

El coprocesador propuesto logra una rapidez de procesamiento apropiada, siendo la tasa de salida igual a 0.691 Gbps para la configuración más básica que consiste en un arreglo de 48 PEs y una frecuencia de operación de 100 MHz. Se mostró en la Tabla 4.4 que el procesamiento más rápido para una misma configuración de parámetros se da cuando el número de filas del elemento estructural es igual al número de PEUs disponibles por cada fila del arreglo sistólico. La rapidez de procesamiento varía de manera inversa al aumento del tamaño del elemento estructural. Por otro lado, al aumentar los valores de los parámetros al sintetizar la descripción realizada, la rapidez de procesamiento aumenta.

Bibliografía

[1] P. Chu, *RTL hardware design using VHDL*. Wiley-Interscience, 2006.

[2] ——, *Embedded SoPC Design with Nios II Processor and VHDL Examples*. Wiley, 2011.

[3] L. Cortés, "A petri net based modeling and verification technique for real-time embedded systems," Master's thesis, Universidad de Linkoping, Linkoping, 2001.

[4] O. Deforges, "A generic systolic processor for real time grayscale morphology," *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, vol. 6, pp. 3331–3334, 2000.

[5] S. Fejes, "A data-driven algorithm and systolic architecture for image morphology," *Image Processing*, vol. 2, pp. 550 – 554, 1994.

[6] H. Kung, "Why systolic architectures?" *Computer*, vol. 15, pp. 37 – 46, 1982.

[7] F. Manriquez, "Caracterización de nódulos de grafito usando morfología matemática," *Acta Microscpica*, vol. 18, no. 2, 2009.

[8] T. G. Morris, "An analogue vlsi morphological image processing circuit," *Electronic letters*, vol. 31, 1995.

[9] D. Nadadur, "Recursive binary dilation and erosion using digital line structuring elements in arbitrary orientations," *Image Processing, IEEE Transactions on*, vol. 9, no. 5, pp. 749–759, 2000.

[10] S. Ongwattanakul, P. Chewputtanagul, D. Jackson, and K. Ricks, "Scalable gigapixels/s binary image morphological operations," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 2, 2003, pp. II–444–II–447 vol.2.

[11] J. Osio, "Desarrollo de algoritmos de procesamiento de imágenes basados en operadores de ventana sobre una fpga." La Plata: Primeras Jornadas de Investigación y Transferencia, 2011.

[12] J. Pastore, "Operadores morfolgicos multiescala y distancia geodsica aplicados a la segmentacin de imgenes de tomografa axial computada," *IEEE Latin America Transactions*, vol. 5, pp. 28–31, 2007.

[13] P. Soille, *Morphological image analysis : principles and applications*. Springer, 1999.

[14] A. Tickle, "Development of morphological operators for field programmable gate arrays," *Journal of Physics: Conference Series*, vol. 76, 2007.

[15] J. Velten, "FPGA-based implementation of variable sized structuring elements for 2d binary morphological operations," in *Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on*, 2002, pp. 309–312.

[16] ——, "High-speed fpga-implementation of multidimensional binary morphological operations," *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 3, pp. 706–709, 2003.

[17] E. Yoshimoto, "Arquitecturas para algoritmos de morfología matemática de imágenes binarias en ahdl," *Iberchip*, 2004.