Aceleración de algoritmos mediante hardware reconfigurable: biblioteca de procesamiento de imágenes para system generator. – La Habana : Instituto Superior Politécnico José Antonio Echeverría (CUJAE), 2012. – Tesis (Maestría).

Dewey: 004 - Ciencia de los computadores.
Registro No.: Maestria1131 CUJAE.

Instituto Superior Politécnico *"José Antonio Echeverría"*

Departamento de Automática y Computación

Tesis para la obtención del grado de Máster en Sistemas Digitales

Aceleración de Algoritmos Mediante Hardware Reconfigurable:

Biblioteca de Procesamiento de Imágenes para System Generator

Ing. Luis Manuel Garcés Socarrás

Tutores

Dr. Santiago Sánchez Solano

Dra. Piedad Brox Jiménez

Dr. Alejandro José Cabrera Sarmiento

La Habana, 2011

Agradecimientos

Agradezco a mi familia por darme la oportunidad de superarme, sobre todo cuando las condiciones atentaron contra ello,
a mi esposa por siempre haber estado ahí y ser la solución a mis problemas,
a mi madre por enseñarme la importancia de seguir adelante,
a mi padre por ser mi meta a seguir y a su esposa por toda la ayuda prestada,
a mis hermanos que aunque lejos nos llevamos en el corazón.
Agradezco además, a mis tutores, Alejandro, por confiar en mí, a Santi por su ayuda incondicional, y a Piedad porque siempre ha estado ahí con una solución viable.
A la AECID por financiar parcialmente esta investigación[1],
a mis amigos, que aunque no los mencione con miedo a olvidar alguno, saben que los aprecio mucho,
a mis estudiantes, pasados, presentes y futuros, que harán de mí un mejor profesional,
a mis compañeros de trabajo de la CUJAE, anteriores profesores y amigos, que me enseñaron parte de lo que sé y a buscar soluciones en lo más recóndito,
al resto del personal de IMSE, amigos todos, cuya ayuda fue necesaria para que esos cuatro meses se sintieran como en casa.
And last, but not the least, I also want to acknowledge Erla and Iain. You know you will always be in my heart.
A todos muchas gracias.
El Autor.

[1]Este trabajo ha sido parcialmente financiado por los proyectos PCI D/024124/09 y D/030769/10 de la Agencia Española de Cooperación Internacional para el Desarrollo (MAEC-AECID). URL: http://www.imse-cnm.csic.es/fortin/

Dedicatoria

"A mi madre, cuya enfermedad no le permitió estar aquí con nosotros"

Resumen

LA presente tesis, titulada *"Aceleración de Algoritmos Mediante Hardware Reconfigurable: Biblioteca de Procesamiento de Imágenes para System Generator"*, desarrolla una biblioteca de procesado de imágenes para el flujo de diseño basado en modelos de System Generator. Este flujo permite la implementación de sistemas de procesado de una manera sencilla y efectiva. El diseño de la biblioteca se basa en el paralelismo de ejecución de los algoritmos de convolución de imágenes y de ordenamiento de píxeles, potenciando la implementación de estos sistemas en arreglos de compuertas programables (FPGA). Además, ésta utiliza métodos de optimización de los algoritmos de procesado desde la parametrización de las versiones genéricas hasta el uso de filtros específicos.

Por último, la efectividad de los bloques de la biblioteca de procesado es comprobada utilizando un sistema de procesado para segmentación de imágenes con XSGImgLib.

Abstract

THIS thesis, entitled *"Acceleration of Algorithms with Reconfigurable Hardware: System Generator's Image Processing Library"*, develops an image processing library for the System Generator's model design flow. This flow allows the implementation of processing systems in a simple and effective way. The design of the library is based on the parallelism of execution of the images' convolution and ranking pixels algorithms, improving the implementation of these algorithms using Field Programmable Gate Arrays (FPGAs). This library also develops optimization methods for generic processing algorithms and specific filters.

Finally, the effectiveness of the library blocks is tested using image's segmentation processing system with XSGImgLib.

Índice general

Índice de figuras

Índice de tablas

Introducción

LOS algoritmos computacionales utilizados para diversos procesos, como son el procesamiento digital de señales, de imágenes, de multimedia, las comunicaciones inalámbricas, la criptografía y las aplicaciones de red, fueron desarrollados en sus inicios sobre procesadores digitales de señales (DSP) o procesadores de propósitos generales (GPP). El avance de las tecnologías de fabricación de circuitos integrados ha permitido el desarrollo de dispositivos lógicos programables de elevadas prestaciones que proporcionan una solución alternativa a la implementación eficiente de este tipo de algoritmos. El aumento significativo del tiempo de cálculo en las nuevas técnicas de procesamiento, puede ser compensado asignando tareas de cálculo intensivas al hardware y explotando el paralelismo de los algoritmos [19].

Arreglos de compuertas programables

Los arreglos de compuertas programables (FPGA) se han convertido en una plataforma opcional para una realización hardware optimizada de algoritmos complejos. El aumento de la complejidad de los algoritmos de procesado requiere una correcta selección del soporte hardware para los diseños. La gran variedad de FPGA existentes en la actualidad permite seleccionar desde dispositivos de alta densidad hasta dispositivos de gran rendimiento, en lugar de costosos sistemas DSP multiprocesadores [19].

Las ventajas de los FPGA con respecto a los demás procesadores para sistemas en un mismo encapsulado (SoC) son notables:

- Posibilitan un alto grado de paralelismo y un aumento de la velocidad de ejecución comparables con potentes GPP.
- Tienen el beneficio del aumento de la velocidad de procesamiento debido al hardware, y la flexibilidad de utilización de software.

- Ofrecen la implementación de sistemas, de forma rápida y poco costosa, con las más avanzadas técnicas de fabricación.

Disímiles arquitecturas y técnicas de programación han evolucionado para proveer mejores diseños que hagan a los FPGA una solución atractiva y económicamente viable para el diseño alternativo de circuitos integrados para aplicaciones específicas (ASIC). Los tres factores claves que juegan un papel importante en el diseño basado en FPGA son [19]:

1. La arquitectura de los FPGA.
2. Las herramientas de diseño electrónico automatizado (EDA).
3. Las diferentes técnicas de diseño de algoritmos.

Herramientas de diseño electrónico automatizado

Las herramientas EDA se clasifican en la categoría de aplicaciones para el diseño electrónico asistido por computadoras. Las mismas funcionan en conjunto con otras herramientas en un flujo de diseño para proyectar, analizar, simular e implementar aplicaciones específicas. Sus inicios datan de los años 60 mostrando un gran auge de crecimiento hasta los años 90. Desde esta fecha a la actualidad, el desarrollo de nuevas herramientas EDA ha decaído, propiciando una madurez en las ya existentes [23].

Tres de las compañías desarrolladoras de estas aplicaciones dominan el 90 % del mercado (Cadence Design Systems, Mentor Graphics y Synopsys Inc.) [24]. Los fabricantes de FPGA, aunque no se encuentran entre los principales productores de herramientas EDA, desarrollan aplicaciones para implementar el flujo de diseño en sus sistemas. Estos programas se enfocan en brindar una solución rápida, optimizada y efectiva para las operaciones de síntesis, colocación, enrutamiento, simulación e implementación de las soluciones hardware planteadas.

Hoy en día los ambientes de diseño de sistemas empotrados se basan en dos tendencias principales: en lenguajes de alto nivel y en modelos visuales. Los entornos de alto nivel son eficientes en las especificaciones del diseño y la verificación del algoritmo pero no pueden ser incluidos fácilmente en un sistema de flujo de datos computarizados. Por el contrario, los diseños basados en modelos, similares a las herramientas esquemáticas tradicionales, presentan bloques, con un gran nivel de abstracción funcional, para la construcción de sistemas permitiendo la integración con otras herramientas de diseño y simulación [28].

Procesamiento digital de imágenes utilizando FPGA

Los algoritmos de procesamiento digital de imágenes (DIP) presentan, en general, una gran carga computacional e imponen serias restricciones temporales para asegurar la operación en tiempo real. Por este motivo la inclusión de algoritmos de procesado de visión en sistemas empotrados con recursos de cálculo limitados, requiere la aplicación de métodos que permitan acelerar su ejecución [4, 32]. La implementación sobre hardware reconfigurable de estos algoritmos surge de la necesidad de cumplir con una serie de requisitos, entre ellos: velocidad de procesamiento, flexibilidad, costo, fiabilidad y tiempo de desarrollo [20, 21].

La utilización de estos dispositivos para la implementación de algoritmos de procesado de imágenes, permite aumentar la velocidad de ejecución con respecto a las soluciones software. No obstante, su uso suele estar condicionado por la disponibilidad de recursos en el dispositivo. La estructura de arreglos de compuertas y registros en paralelo de los FPGA hacen a las mismas una opción viable para explotar el paralelismo de datos de las imágenes o tramas de vídeos. Éstas pueden ser utilizadas para realizar operaciones completas o para preprocesar los datos antes de enviarlos a un DSP estándar o a un microprocesador [12].

Diseño teórico

La utilización de herramientas EDA, integradas con MATLAB/Simulink, para la realización de sistemas empotrados ha permitido el desarrollo de implementaciones hardware de una manera sencilla y económica en cuestiones de tiempo de diseño. La actual parametrización de los bloques básicos brindados por los diseñadores permite el desarrollo de sistemas con una adecuada optimización, aunque no al nivel de los implementados con técnicas convencionales (HDL). El desarrollo de nuevas bibliotecas con mayores posibilidades de configuración permite elevar las potencialidades de los diseños basados en modelos.

System Generator (XSG), herramienta provista por Xilinx e integrada con MATLAB/Simulink, permite el diseño de sistemas basados en modelos [22]. XSG presenta, entre otras, una biblioteca de bloques de procesamiento de imágenes basada en convoluciones [37], pero con posibilidades de configuración limitadas y necesita de otros algoritmos de procesado.

Problema científico

Dado el planteamiento anterior se llega al siguiente problema científico: No existe una biblioteca optimizada y parametrizable para el diseño de sistemas empotrados reconfigurables para el procesamiento de imágenes con System Generator.

Del problema científico se deriva la hipótesis: El desarrollo de una biblioteca de procesamiento de imágenes en escala de grises para System Generator, optimizada, y con mayor grado de configurabilidad que la brindada por Xilinx, permitiría el diseño de sistemas empotrados aumentando la frecuencia de ejecución y disminuyendo la utilización de recursos.

Objetivo

El objetivo de la investigación se centra en: Desarrollar una biblioteca de procesamiento de imágenes en escala de grises, con funciones básicas, parametrizables y optimizadas en velocidad o consumo de recursos, que permita el diseño de sistemas de procesamiento de imágenes para sistemas empotrados utilizando el flujo de diseño de MATLAB/Simulik-XSG-ISE.

La biblioteca XSGImgLib, resultado de esta investigación, se compone de bloques de procesamiento de imágenes para ventanas de 3×3 y 5×5 distribuidas en procesamiento lineal, procesamiento no lineal, operadores morfológicos y otros bloques, los cuales se describen en capítulos posteriores (ver Anexo 1).

Para el cumplimiento del objetivo se trazan las siguientes tareas:

1. Clasificar los algoritmos de procesamiento de imágenes básicos en cuanto a la posibilidad de implementarlos sobre hardware reconfigurable.
2. Desarrollar una biblioteca de procesamiento de imágenes utilizando el flujo de diseño basado en modelos.
3. Evaluar los bloques de procesamiento desarrollados.
4. Comparar los elementos de la biblioteca con otros ya existentes.
5. Demostrar la utilización de los bloques de la biblioteca en la implementación de un sistema de procesamiento de imágenes.

Composición del trabajo

El trabajo se compone de introducción, cinco capítulos, conclusiones, recomendaciones, bibliografía y anexos.

El Capítulo 1: *"Procesamiento digital de imágenes"* aborda temas básicos del procesamiento digital de imágenes, su definición matemática y aspectos de la representación de la misma en un modelo de color. Además, detalla el estado del arte del procesamiento digital de imágenes en sistemas empotrados y la utilización de herramientas EDA en el desarrollo de sistemas de procesado para FPGA.

El Capítulo 2: *"Filtros de procesado lineal"* describe el procesamiento de imágenes basado en algoritmos de convolución. Primeramente, aborda la convolución en dos dimensiones como operación matemática, para luego, presentar diferentes arquitecturas de bloques de convolución de procesado de imágenes, implementadas en la biblioteca XSGImgLib. El capítulo concluye con la explicación de la parametrización de los bloques de convolución desarrollados y su comparación. Además, se analiza el bloque de convolución de System Generator.

Los bloques de procesado no lineal de la biblioteca XSGImgLib son expuestos en el Capítulo 3: *"Filtros de procesado no lineal"*. En éste se explica el funcionamiento de los bloques basados en ordenamiento y el bloque de umbralización. Además, se describen las diferentes arquitecturas hardware de estos bloques, realizando comparaciones entre las mismas. El capítulo culmina con el análisis de un bloque de ordenamiento basado en arquitecturas similares pero utilizando otro flujo de diseño.

El Capítulo 4: *"Operadores morfológicos y bloques de control"* aborda los operadores morfológicos y sus arquitecturas, haciendo énfasis en el consumo de recursos de cada uno de ellos. También se explican los módulos que forman parte de la paleta de control de la biblioteca, necesarios para el desarrollo de sistemas con XSGImgLib.

Por último, el Capítulo 5: *"Procesado de imágenes utilizando XSGImgLib"* describe un ejemplo de aplicación de la biblioteca XSGImgLib para segmentación de imágenes. El mismo tiene en cuenta la simulación funcional del algoritmo, detectando la ocurrencia de errores en cada etapa de procesado, y la co-simulación hardware del sistema utilizando una placa de desarrollo. También se muestran imágenes de los resultados del procesamiento así como el consumo de recursos del sistema.

Capítulo 1
Procesamiento digital de imágenes

EL procesamiento digital de imágenes constituye, hoy en día, un campo de amplias aplicaciones. Entre las principales ramas que demandan su uso se encuentran la salud, la seguridad, el entretenimiento y las aplicaciones militares y aeronáuticas. Dado su amplio desarrollo, el procesamiento digital de imágenes requiere de nuevos diseños y configuraciones para optimizar su ejecución.

En este capítulo se introducen aspectos básicos del procesamiento digital de imágenes así como el estado del arte de los algoritmos utilizados en el mismo. Además, se aborda la utilización de las herramientas de diseño automatizado para el desarrollo de sistemas empotrados de procesamiento de imágenes utilizando el flujo de diseño basado en modelos.

1.1. Introducción al procesamiento digital de imágenes

Una imagen puede ser definida por una función bidimensional $A(x, y)$, como se muestra en la Figura 1.1, donde x e y son coordenadas espaciales. La amplitud de A en el par de coordenadas (x, y) se conoce como intensidad de color o nivel de gris de la imagen en el punto. Cuando x, y y el valor de la amplitud de A son valores finitos (cantidades discretas) se trabaja con una imagen digital [9].

La función $A(x, y)$ de una imagen digital es representada en su forma compacta por una matriz de $m \times n$ píxeles, como muestra la Ecuación 1.1, o en su versión más tradicional,

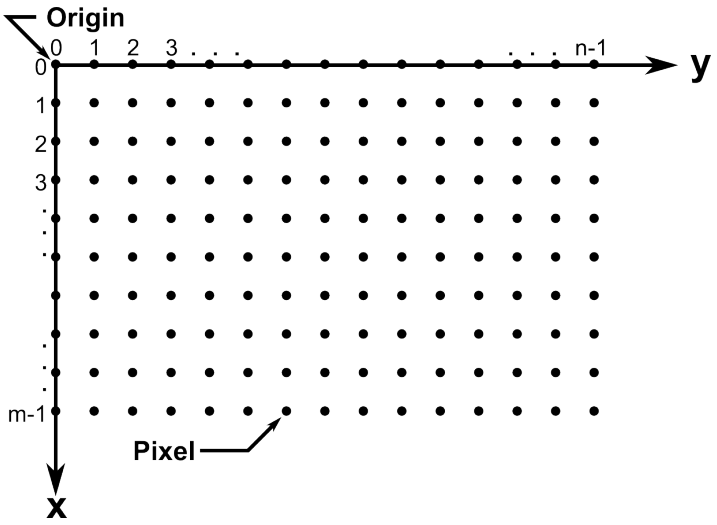


Figura 1.1: Representación de una imagen [9].

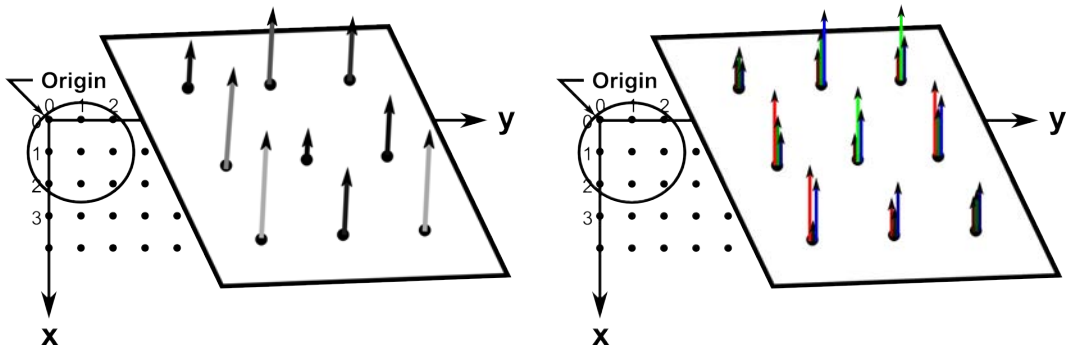
expuesta en la Ecuación 1.2.

$$A(x,y) = \begin{bmatrix} A(0,0) & A(0,1) & \dots & A(0,n-1) \\ A(1,0) & A(1,1) & \dots & A(1,n-1) \\ \vdots & \vdots & \ddots & \vdots \\ A(m-1,0) & A(m-1,1) & \dots & A(m-1,n-1) \end{bmatrix}. \tag{1.1}$$

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & \dots & a_{m-1,n-1} \end{bmatrix}. \tag{1.2}$$

1.2. Tipos de imágenes

La representación de la amplitud de la función A depende de la composición de la misma. Para imágenes en escala de grises la digitalización es función del tamaño de la imagen (m y n) y del nivel en la escala de grises para cada punto (l) (Figura 1.2a). No existen requerimientos específicos para los valores de m y n, sólo que deben ser números enteros



(a) Escala de grises. **(b)** RGB.

Figura 1.2: Modelos de representación de una imagen.

positivos, mientras que el nivel en la escala de grises es usualmente representado por un valor en potencia de 2 (Ecuación 1.3) [9].

$$l = 2^k. \tag{1.3}$$

Las características principales para distinguir un color de otro vienen dadas por la saturación, el brillo y el color. El brillo es la noción cromática de la intensidad, el color representa la longitud de onda (λ) del color dominante y la saturación se refiere a la cantidad de luz blanca mezclada con el color. La unión del color y la saturación se conoce como cromaticidad, por lo que un color puede ser representado mediante la cromaticidad y el brillo [9].

Las imágenes en colores utilizan la representación de los píxeles en un modelo de color, ubicando un punto de color en un sistema de coordenadas. Uno de los modelos más utilizados para la representación de imágenes en colores es el RGB[1] utilizado en los visualizadores. El modelo RGB mezcla los espectros de los colores primarios rojo, verde y azul en diversas intensidades para formar el color, como muestra la Figura 1.2b [9].

[1]Modelo de color Rojo, Verde y Azul (*Red-Green-Blue*).

1.3. Procesamiento digital de imágenes

Los métodos de procesamiento digital de imágenes permiten realizar diferentes tareas de acondicionamiento y mejora de la información pictórica con objeto de facilitar, tanto la interpretación humana, como el procesado, almacenamiento y transmisión de imágenes mediante sistemas de percepción autónomos [9]. Éstos se basan en la transformación de todos los puntos de la imagen mediante la aplicación de una función matemática, expuesta en la Ecuación 1.4, donde i va desde $x - rw$ hasta $x + rw$ y j desde $y - rw$ hasta $y + rw$, siendo rw el radio de la ventana de píxeles.

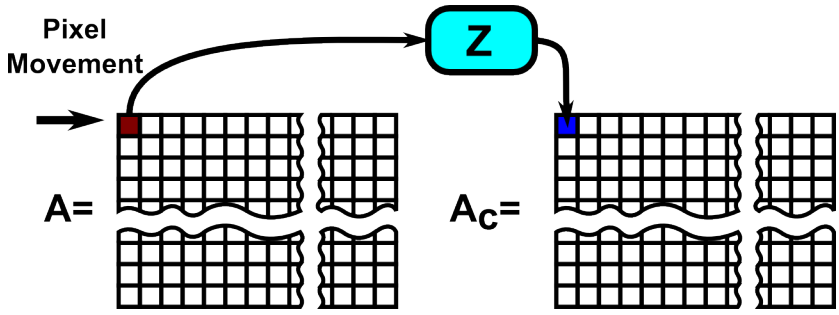
$$A_c(x - rw, y - rw) = ZA(i, j). \tag{1.4}$$

El procesamiento se realiza aplicando una transformación matemática (Z), que define los cambios a realizar en la imagen original, utilizando uno o varios píxeles de acuerdo con el tipo de procesado. Todos los píxeles de la imagen son recorridos aplicando la misma transformación. La cantidad de píxeles a utilizar en el procesamiento está definida por el radio de píxeles en la vecindad a analizar. Cuando el radio de la vecindad es cero ($rw = 0$) el procesamiento es puntual (Figura 1.3a) sobre el píxel (x, y), sin embargo si el radio de la vecindad es un número natural distinto de cero el procesamiento se realiza sobre el píxel (x, y) teniendo en cuenta los píxeles circundantes (Figura 1.3b) [4].

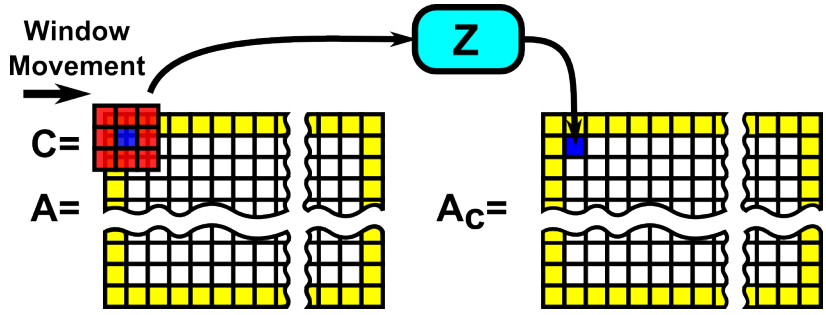
Las técnicas de procesado de imágenes se clasifican, de acuerdo a la complejidad de la operación a realizar, en métodos de bajo, medio y alto nivel (Figura 1.4). Entre las funciones principales de los algoritmos de procesado de imágenes están la adquisición de la imagen, el preprocesamiento (mejoramiento de la imagen, eliminación de ruido, etc. [22]), la segmentación (detección de contornos y umbralización), la extracción de características, y la interpretación de la información.

1.3.1. Arquitecturas de procesamiento de imágenes para sistemas autónomos reconfigurables

Muchas arquitecturas de procesamiento de imágenes y vídeos sobre hardware reconfigurable han sido planteadas por numerosos autores, en aras de utilizar el paralelismo de los



(a) Puntual.



(b) Vecindad.

Figura 1.3: Tipos de procesado de imágenes.

diferentes algoritmos y la velocidad de ejecución con respecto a las soluciones software [4]. Las últimas innovaciones de la ciencia, como la televisión digital y el cine, se centran en la rápida evolución del procesamiento de imágenes y vídeo. Entre estas técnicas se pueden contar algoritmos avanzados de compresión, de vídeo inteligente y de seguimiento, entre otras. Los nuevos estándares de la televisión de alta definición (HD) representan un aumento de seis veces la cantidad de información utilizada en la televisión de definición estándar (SD) [2]. Los sistemas de vigilancia cambian a los nuevos estándares aumentando la resolución y por ende la información a procesar. Las aplicaciones militares, médicas y de visión computarizada necesitan de este aumento de la información para la obtención de mejores resultados.

Benkrid *et al.* plantean en [3] el diseño de arquitecturas de convolución para hardware reconfigurable optimizando el consumo de recursos. El paralelismo brindado por la definición matemática del algoritmo de convolución para dos dimensiones, unido a las características de los diferentes núcleos ya definidos, permiten el desarrollo de arquitecturas específicas eliminando bloques de multiplicación o sustituyéndolos por rotaciones y sumas.

Otro trabajo del autor [4] desarrolla un entorno para la implementación de sistemas de

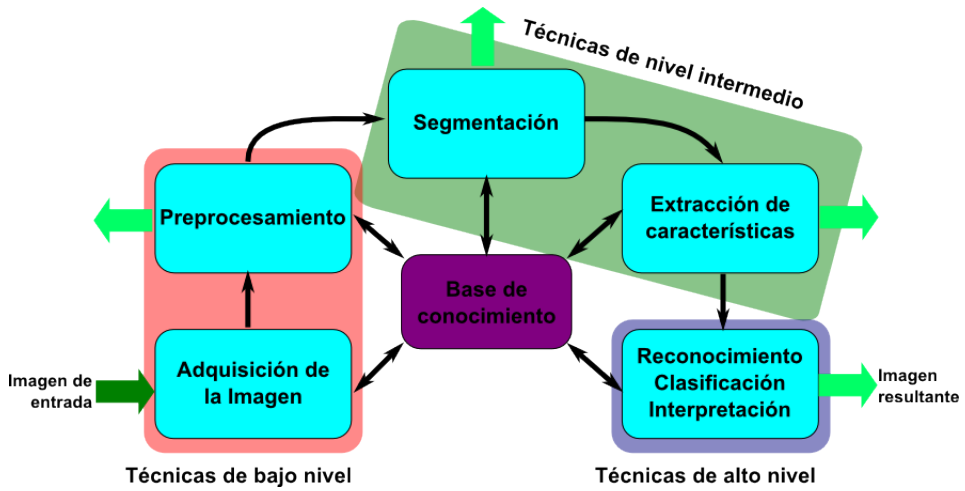


Figura 1.4: Clasificación de las técnicas de procesado de imágenes.

procesado de imágenes basados en la optimización de bloques ya definidos y orientados a aplicaciones. La configuración de los bloques hardware hace que los diseños mantengan un nivel de eficiencia y rendimiento aceptable. Muchos algoritmos de procesado de imágenes pueden ser divididos en un flujo de tareas a realizar. Esta característica permite la utilización de esbozos ya diseñados para implementar un sistema de procesamiento más complejo [4].

En [18] se plantea una arquitectura para procesamiento espacial no lineal de un filtro de mediana, basado en el ordenamiento de los píxeles. El diseño, desarrollado en lenguaje de descripción de hardware, posee un bloque de ordenamiento, utilizando un comparador de dos entradas para obtener el píxel mayor y el menor, y un módulo serie de comunicación con la computadora. El algoritmo ordena los valores de los píxeles en la ventana y activa la salida del píxel central luego de varios ciclos de reloj, consumiendo gran cantidad de recursos y períodos de ejecución [18, 15]. El aumento del tamaño de la ventana utilizando este algoritmo requiere un mayor tiempo de ejecución y una mayor cantidad de recursos.

Nelson dedica su tesis de Máster en Ciencias [15] al estudio de algoritmos software y hardware de procesado lineal, no lineal y morfológico. Las pruebas a los diseños son realizadas en MATLAB, para luego ser convertidos a lenguaje de descripción de hardware. Estos algoritmos son integrados en un ambiente de desarrollo basado en modelos llamado ACS utilizando la herramienta Graphical Model Editor (GME) [15].

La utilización de la plataforma GME también se pone de manifiesto en [16] para la creación de un sistema automático de reconocimiento de objetivos (ATR) para misiles. Un ATR debe

cumplir con varios requerimientos físicos como: tamaño, consumo de potencia y resistencia al calor. Dada la alta carga computacional de los algoritmos implementados, junto con la necesidad de precisión de estos sistemas, el procesamiento en tiempo real y la posibilidad de reconfiguración del mismo son características imprescindibles para la puesta en funcionamiento del ATR. Los resultados del trabajo corroboran la fiabilidad de la utilización de modelos para el desarrollo de sistemas de alta precisión [16].

Wnuk, en [32] realiza un análisis de arquitecturas eficientes para desarrollar sobre hardware. Para el desarrollo de bloques de convolución, el autor plantea dos posibles diseños: la utilización de la definición matemática de la función de convolución de dos dimensiones y el uso de bloques de multiplicación y acumulado (MAC) [32]. Los diseños difieren en la necesidad del uso de bloques de memorias dedicadas (BRAM) y la reducción de los bloques de multiplicación para el diseño con bloques MAC, mientras que el basado en la definición matemática reduce el tiempo de ejecución y aumenta la cantidad de recursos lógicos utilizados. Además el artículo aborda el diseño de un caso especial, planteado por Golston en 1997, para el diseño de un filtro de mediana, utilizando como base un bloque de ordenamiento de tres entradas. Este diseño sólo es viable para una vecindad de 3×3 píxeles por lo que es necesaria la utilización de otras arquitecturas para vecindades de píxeles mayores.

Chakrabarti plantea en [8] una solución al problema anteriormente planteado, utilizando filtros de ordenamiento genéricos recursivos y no recursivos [8, 26]. Dada las limitaciones de recursos de los dispositivos hardware, el caso de interés de este trabajo es el diseño no recursivo del mismo. El diseño de un filtro no recursivo de ordenamiento de dos dimensiones se basa en la versión de una dimensión. El filtro de dos dimensiones de $k \times k$ elementos se compone de dos tipos de procesadores, los que analizan las nuevas muestras y detectan la posición de los elementos antiguos (k procesadores), y los que almacenan las muestras antiguas manteniendo un registro del estado de la posición de las muestras en el sistema ($k^2 - k$ procesadores). Esta arquitectura, la cual es abordada en detalles más adelante, elimina la necesidad de realizar la comparación de todos los elementos dentro de la ventana, al desplazar la misma por la imagen [8].

El diseño de sistemas parcialmente reconfigurables se presenta en [5] como una opción para la implementación de algoritmos de procesado de imágenes y vídeos. La reconfiguración parcial permite acelerar diversos algoritmos, compartiendo los recursos del FPGA sin necesidad de detener el sistema de procesamiento. En este sistema, un filtro está en ejecución

mientras que los demás permanecen inactivos, esperando la selección del próximo filtro a ejecutar por medio del procesador [5].

La aplicación de los dispositivos empotrados en sistemas de monitoreo y seguimiento permite el desarrollo de aplicaciones modulares, parcialmente independientes y con un bajo consumo de potencia [6]. En [6] se describe la arquitectura de un sistema de seguimiento y detección inteligente de tráfico utilizando DSP. El sistemas se divide en tres unidades con tareas específicas, la unidad de sensado encargada del manejo de la cámara para la detección de los datos, la unidad de procesamiento dedicada a la conversión de los mismos, el análisis de la información y la detección de eventos del sistema, y la unidad de comunicación destinada al intercambio de información con el controlador maestro del sistema [6].

Los FPGA tienen un gran campo de acción en la industria para el desarrollo de aplicaciones específicas. En [10] se desarrolla un sistema de procesamiento de vídeo para la detección de irregularidades en la superficie de productos terminados. El sistema está compuesto por una cámara de vídeo, un microcontrolador, una memoria $EEPROM$ y un FPGA de Xilinx encargado del procesamiento. Los algoritmos de procesado se basan en la umbralización de la imagen obtenida desde la cámara y la detección de diferencias entre los píxeles para analizar la superficie. Este sistema utiliza un microcontrolador para el manejo del módulo de adquisición y la comunicación con un sistema de mando vía RS-232 [10]. Actualmente es posible incluir estas funciones dentro del mismo FPGA.

1.3.2. Procesamiento de imágenes con System Generator

Un modo de acelerar el proceso de diseño de sistemas sobre FPGA, manteniendo el nivel de eficiencia, es el desarrollo de bloques optimizados y parametrizables en función de las necesidades del usuario [3]. System Generator permite el desarrollo e implementación de sistemas empotrados sin necesidad de conocer a fondo la programación en lenguajes de descripción de hardware, realizando los procesos de síntesis, implementación y descarga en placa automáticamente desde un modelo en Simulink [20–22]. Esta ventaja es a su vez el principal punto débil de la herramienta, ya que al existir una abstracción del usuario a la aritmética utilizada, se necesita lograr una adecuada parametrización de los modelos para brindar una solución viable a las necesidades planteadas.

El flujo de diseño con System Generator, como bien antes se menciona, utiliza la metodología de diseño basada en modelos de Simulink. Las especificaciones del mismo permite el uso de aritmética de punto flotante, ajustando los detalles de implementación de hardware para los dispositivos de Xilinx. Además, éste posibilita la creación de un conjunto de pruebas para la utilización de simuladores funcionales y temporales para el diseño (ModelSim[2] o Xilinx ISE Simulator[3]).

La posibilidad de simulación del diseño desarrollado en software, con MATLAB/Simulink, y su verificación en hardware utilizando co-simulación (Figura 1.5) son otras de las ventajas de System Generator sobre el diseño convencional de sistemas [21]. Para la opción de co-simulación, se genera un modelo de simulación para hardware del diseño realizado desde el entorno de desarrollo. Este modelo presenta la parte del procesado elaborado para el dispositivo físico, permitiendo el diseño de un nuevo sistema o la reutilización de los bloques de entrada y de salida del antiguo, para comprobar la ejecución del algoritmo. Al efectuar la simulación del modelo hardware, previamente se realizan los procesos de síntesis, posicionamiento, ruteo y descarga en placa de algoritmo de procesado, para luego obtener los datos desde el entorno de desarrollo (MATLAB/Simulink) y enviar los mismos hacia el FPGA. Éste realiza la ejecución del algoritmo configurado y retorna el resultado al entorno de desarrollo. La utilización de esta técnica, además de la comprobación del diseño desde el dispositivo físico y la posibilidad de entrada y recepción de datos del entorno de desarrollo, permite un incremento del rendimiento de hasta 1000 veces el rendimiento de la simulación del sistema. Esta cifra depende tanto de la placa de desarrollo utilizada, como de la interfaz de conexión con la misma, siendo JTAG[4], USB, Ethernet y punto a punto las conexiones disponibles [36].

La biblioteca de módulos disponible en System Generator proporciona una serie de bloques básicos para el desarrollo de sistemas complejos en los FPGA de Xilinx [27, 28]. El bloque de convolución genérico, provisto por Xilinx, implementa las operaciones de multiplicación y acumulación de los píxeles de la imagen con un núcleo o kernel de convolución de 5×5 [37]. El diseño del bloque de procesado se compone de cinco filtros de respuesta finita al impulso (FIR) de tipo MAC, a su vez compuestos por multiplicadores y bloques de memoria dedicados [20].

[2]Simulador de diferentes lenguajes de programación hardware fabricado por Mentor Graphics.
[3]Simulador provisto por Xilinx e incluido en la plataforma Xilinx ISE Design Suite.
[4]Nombre común utilizado para la norma IEEE 1149.1 para el *Standard Test Access Port and Boundary-Scan Architecture* utilizada para analizar circuitos integrados.

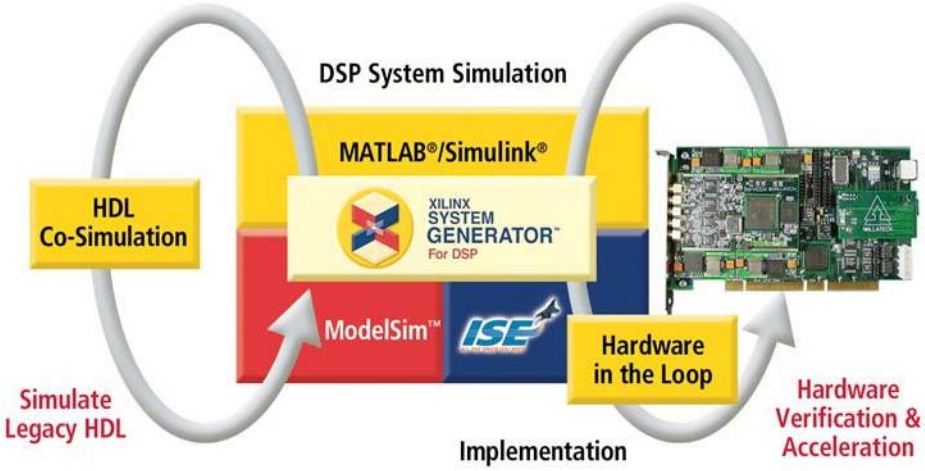


Figura 1.5: Flujo de diseño con System Generator [33].

Otro de los estudios sobre la utilización de System Generator en el procesamiento de imágenes se plantea en [22]. Este artículo presenta una descripción del tipo de procesado de imágenes, así como diferentes arquitecturas implementadas con System Generator para mejorar el contraste, el brillo y la detección de bordes, entre otros tipos de procesamiento.

La alternativa propuesta en [27] plantea el desarrollo de sistemas de procesado de imágenes utilizando bloques hardware (HDL y XSG) y bloques software (MATLAB y OpenCV[5]) [27]. La utilización de bloques híbridos con dos niveles de abstracción, distintos al provisto por XSG, permite la portabilidad entre diversas plataformas mediante modelos de las distintas interfaces de conexión. La biblioteca desarrollada haciendo uso el flujo de diseño planteado, mantiene el compromiso entre la eficiencia y la flexibilidad, posibilitando la configuración de los parámetros de los bloques a las necesidades del usuario. Estos valores de configuración permiten además la selección de la precisión de las unidades de cálculo, así como la latencia y el período de muestreo [27].

Las arquitecturas de procesamiento de imágenes desarrolladas en System Generator son aplicadas al procesado de vídeo realizando algunas modificaciones que permitan el manejo de las señales de sincronismo. Saidani *et al.* plantean en [21] la utilización de flujo de diseño de XSG para la implementación de un sistema de conversión de espacio de colores RGB - YCbCr[6]. Este diseño es utilizado en la transformación de la señal RGB de la salida

[5] Biblioteca libre de visión artificial originalmente desarrollada por Intel.
[6] Modelo de representación digital del color donde: Y transporta la luminancia (brillo), Cb la direfencia entre la componente azul y la luminancia, y Cr la diferencia entre la componente roja y la luminancia.

de una tarjeta de vídeo VGA a la entrada YCbCr de un televisor. El diseño permite una interfaz gráfica amigable para el desarrollo del sistema de procesamiento posibilitando su comprobación por medio de la co-simulación hardware [21].

Szedo *et al.* publican una nota de aplicación para Xilinx [26] donde utilizan la arquitectura no recursiva de Chakrabarti [8] en la implementación de un sistema ampliamente configurable de reducción de ruidos para señales de vídeo. El diseño, realizado sobre lenguaje de descripción de hardware e incluido en Simulink utilizando un bloque de caja negra, permite una amplia gama de selección de parámetros, la realización de la comprobación del sistema utilizando simulación y co-simulación, y el desarrollo de un sistema eficiente y optimizado. El sistema es capaz de procesar imágenes en colores realizando una conversión de color RGB a YCbCr y utiliza bloques de memoria para almacenar la trama a procesar [26]. El uso de BRAM implica una limitación en la resolución de las tramas a analizar, por lo que se buscan soluciones alternativas, analizadas posteriormente.

1.3.3. Diseño de sistemas de procesado de imágenes con XSG

System Generator es un ambiente muy utilizado para diseñar diversas aplicaciones específicas; entre ellas se encuentra el procesamiento de imágenes. En los sistemas de procesado de imágenes la entrada de datos se realiza recepcionando el valor de un píxel en cada ciclo de reloj, dada por la secuencia inherente al proceso de transmisión de la imagen. Además, la posibilidad que brinda XSG en la visualización del resultado del procesado, utilizando la adquisición de datos desde Simulink y la representación de la salida desde el espacio de trabajo de MATLAB, permite la comprobación del mismo mediante la percepción visual y los algoritmos matemáticos realizados en software. Un sistema básico de procesado de imágenes, como el mostrado en la Figura 1.6, se compone de un subsistema de entrada, un subsistema de salida, y entre ellos el subsistema de procesado. Este último es desarrollado utilizando los bloques básicos de la biblioteca de System Generator, siendo además la sección del diseño a implementar en hardware.

El sistema detallado a continuación es tomado como patrón para el diseño, simulación, co-simulación y comprobación de los bloques de procesado descritos en capítulos posteriores. En los diseños que se plantean, la imagen es analizada previamente en MATLAB obteniendo parámetros como las dimensiones de la misma, la conversión a escala de grises (en caso necesario) y la creación de un vector de entrada con los datos de la imagen serie (para

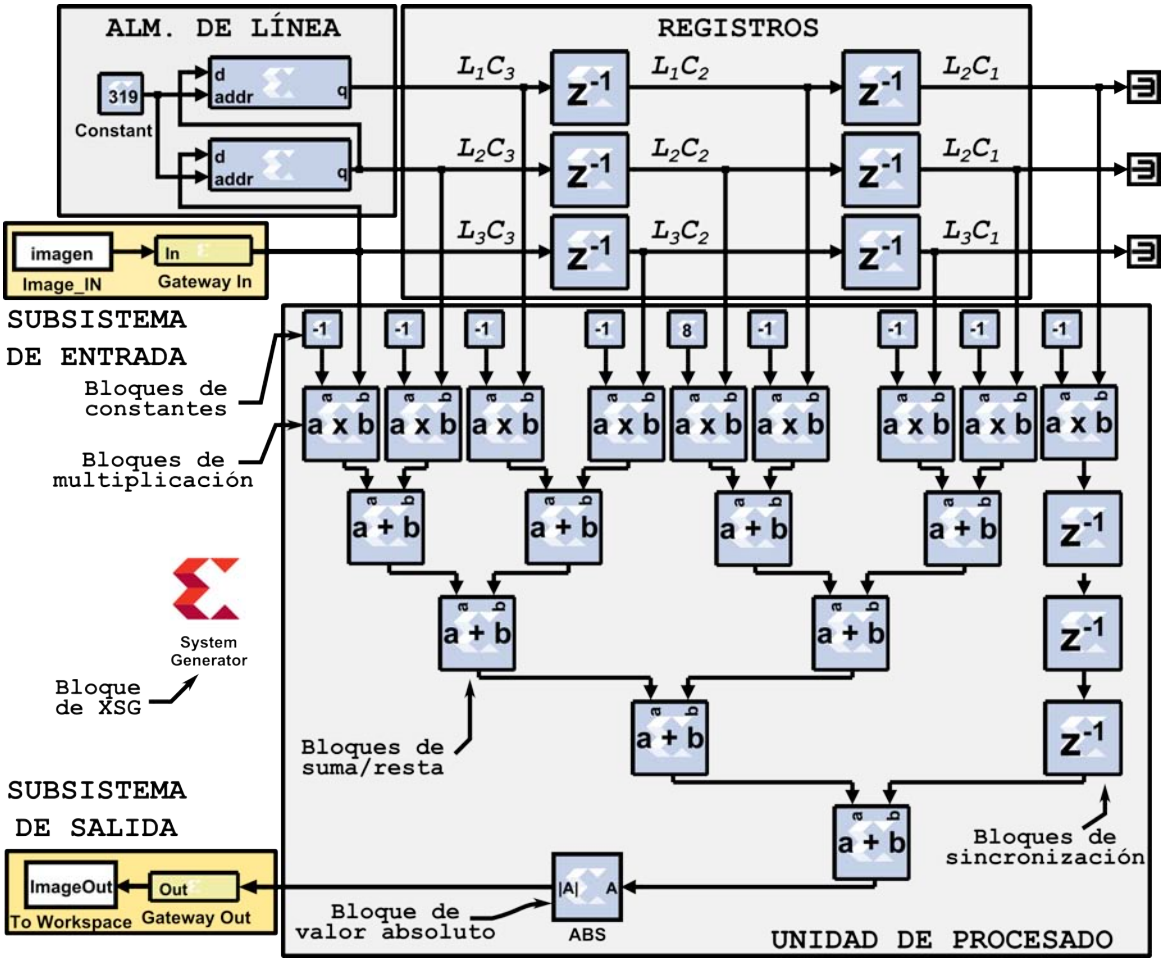


Figura 1.6: Sistema de procesado de imágenes con XSG.

cumplir con los estándares de transmisión). Esto último permite la utilización del subsistema de procesado para aplicaciones con entrada de datos mediante cámaras u otros dispositivos físicos, agregando el trabajo con las señales de sincronismo.

En todo sistema de diseño para XSG debe incluirse el bloque de procesado de System Generator (Figura 6.1). Éste permite que todos los bloques de XSG sean compilados por las herramientas de Xilinx para la simulación del diseño, además de contener varias funciones en su pestaña de configuración, entre ellas: opciones de compilación, opciones de temporización y opciones de simulación. Estos parámetros son detallados en el Anexo 3.

Subsistema de entrada y salida de datos

Los subsistemas de entrada y salida de datos son los encargados del intercambio de información entre el espacio de trabajo de MATLAB y el algoritmo de procesado, tanto para el desarrollado en hardware (co-simulación HW), como para la simulación. El sistema de entrada está delimitado por uno o varios bloques *Gateway In*[7] (dependiendo de la cantidad de entradas del diseño), delante de los cuales se encuentran estructuras o bloques de Simulink para la entrada de información al sistema. En cambio el subsistema de salida comienza con uno o varios bloques *Gateway Out*[8] seguidos de bloques de Simulink para la recepción o el procesado de los resultados por software.

La puerta de entrada presenta parámetros configurables como son: la cantidad de bits para la representación de los valores de entrada al subsistema de procesado, los métodos para la limitación de la misma y los parámetros de implementación. La puerta de salida permite la configuración de los parámetros de salida del sistema hardware, así como las restricciones del mismo.

Para la implementación de un sistema de procesado de imágenes en escalas de grises, donde los valores de la entrada presentan 256 niveles distintos sin signo, ni parte decimal, la puerta de entrada es configurada para valores enteros de 8 bits sin signo. El método de limitación de la entrada trunca y satura los valores de los píxeles para asegurar que los niveles de grises no sobrepasen el nivel máximo. La puerta de salida mantiene la configuración por defecto.

Subsistema de procesado

El subsistema de procesado está limitado por los subsistemas de entrada y el de salida. Este bloque es el más importante del diseño ya que, a partir de éste, se obtienen los parámetros de configuración de las unidades lógicas y específicas del dispositivo físico, para lograr la aplicación deseada.

Para el procesado de imágenes basado en arquitecturas paralelas, el subsistema de procesado está compuesto por una unidad de paralelización de los píxeles de entrada por cada canal

[7] Puerta de entrada.
[8] Puerta de salida.

de entrada, condicionada por el tamaño de la ventana en la unidad de procesado. Además, puede presentar otras entradas de señales necesarias para aplicaciones específicas.

Unidad de paralelización de la información

En cualquier sistema de procesado de imágenes y vídeo la imagen de entrada es serializada píxel a píxel desde el sistema de transmisión (subsistema de entrada en la Figura 1.6) al sistema de procesado (subsistema de procesado en la Figura 1.6) y visualización (subsistema de salida en la Figura 1.6). Para explotar las posibilidades de paralelismo de ejecución es necesario que la información de la imagen llegue de forma paralela al bloque de procesado. Por ello se requiere la utilización de bloques que permitan convertir el flujo de información serie a un flujo paralelo. Los bloques almacenadores de línea (*Line Buffers*) y los registros (*Registers*) son los elementos que, aunque no forman parte de la unidad de procesado, permiten esta conversión y así el paralelismo del algoritmo [1, 5, 9, 18, 32]. Los valores paralelos de los píxeles de la imagen en la ventana son las señales L_aC_b (con $1 \leq a \leq h$, $0 \leq b \leq h$ y h es el tamaño de la ventana) siendo L_h y C_h la fila y columna más recientes respectivamente. Estos bloques se describen en la Sección 4.2.

Los bloques almacenadores de línea se construyen mediante bloques de registros de desplazamiento direccionables los cuales presentan como parámetros la cantidad de etapas de demora, y la selección de la etapa a la salida. Los registros se componen de bloques básicos de XSG, demorando la entrada un período de reloj. Las propiedades de estos bloques se describen en el Anexo 3.

Para el sistema que se describe, los parámetros de los almacenadores de línea se configuran de acuerdo a la cantidad de píxeles de la imagen en el eje horizontal, habilitando la salida de la última posición.

Unidad de procesado

La unidad de procesado, mostrada en la Figura 1.6, es la encargada de realizar el algoritmo matemático sobre la imagen. Ésta está compuesta generalmente por bloques de constantes, bloques de operaciones matemáticas, bloques de sincronismo, y un bloque del cálculo del valor absoluto. Este último es necesario para la conversión de la salida, asegurando que los

valores resultantes cumplan los requerimientos para imágenes en escalas de grises con 2^8 niveles (enteros positivos sin signo).

Los bloques de constantes (Figura 6.14) almacenan los valores del kernel seleccionado. El mismo es introducido manualmente o seleccionado mediante la lista desplegable de la ventana de configuración del bloque de procesado. Los bloques de constantes sólo son necesarios cuando el bloque de procesado es genérico basado en convolución o en operadores morfológicos, para almacenar los coeficientes del kernel o del elemento estructural respectivamente. La configuración de estos bloques permiten la selección de la representación del valor, así como otros parámetros abordados en el Anexo 3. La selección del kernel, en la pestaña de configuración del bloque de procesado, realiza la normalización del mismo y parametriza los valores de cada bloque de constante. Éste aspecto es descrito en la Sección 2.1.

Las operaciones matemáticas del algoritmo son implementadas con bloques matemáticos de XSG, presentando dos entradas generalmente. Estos bloques dependen del diseño desarrollado permitiendo la configuración de varios parámetros (la representación de la salida, la latencia, entre otos expuestos en el Anexo 3). Los bloques de multiplicación (Figura 6.4) pueden utilizar bloques dedicados de la placa de desarrollo o recursos lógicos. Los bloques dedicados permiten optimizar la frecuencia de la operación, mientras que el uso de bloques lógicos posibilita el ahorro de los recursos específicos.

Toda operación matemática, incluso con bloques de multiplicación dedicados, necesita de una cierta cantidad de ciclos de reloj para la obtención del resultado. Esta latencia de la operación implica el uso de mayor cantidad de recursos lógicos a costa de una mejor frecuencia de trabajo. Éstos parámetros se describen en el Anexo 3.

Dado a que la mayoría de los bloques de procesado presentan sólo dos entradas, al querer realizar operaciones con mayor cantidad de entradas es necesario implementar operadores en cascada. Cuando la cantidad de entradas es impar es imprescindible el uso de bloques de registro como unidades de sincronismo de las operaciones. Ésto permite que cada operación se realice con los valores que corresponden sin agregar algún error al procesado.

Comprobación del sistema de procesado

Una vez concluido el subsistema de procesado, conectando los subsistemas de entrada y salida al espacio de trabajo de MATLAB, se procede a la comprobación del diseño reali-

zado. Durante la simulación el subsistema de entrada recibe la información de la imagen, previamente serializada, y es ajustada por la puerta de entrada antes de enviársela al subsistema de procesado. Ya en éste, los almacenadores de línea acumulan $2 \times rw$ filas de la imagen, mientras que los registros lo hacen con $2 \times rw$ columnas (donde rw es el radio de la ventana), para paralelizar los valores a enviar a la unidad de procesado. Los píxeles en paralelo pasan a la unidad de procesado y ésta entrega el resultado al subsistema de salida de forma serie. Este último devuelve la salida a MATLAB, el cual se encarga de almacenarla, procesarla y visualizarla por software.

Dada a las demoras inherentes en el sistema de procesado, por el llenado de los almacenadores de línea, los registros y el tiempo necesario para el procesado, existe una latencia en todo el sistema de procesado denotada como τ_s. Esta latencia implica que el primer píxel válido no estará en la salida hasta pasado este tiempo, por lo que en la imagen de salida se observan píxeles sin información (píxeles negros). Al salir el primer píxel válido, éste se encuentra en otra posición en la imagen, ocurriendo un desplazamiento de los píxeles en la misma. Este problema es corregido en los sistemas de visualización mediante el uso de las señales de sincronismo de vídeo, y por software eliminando la información no válida. Estos aspectos son abordados en capítulos posteriores.

Para la co-simulación hardware, se selecciona esta opción en el bloque de XSG y se realiza la compilación del subsistema de procesado. En la configuración debe seleccionarse la placa de desarrollo y el tipo de conexión a utilizar con ésta. Una vez concluida la compilación, el sistema crea un nuevo bloque (*Model hwcosim* en la Figura 1.7) que debe sustituir al subsistema de procesado. Al ejecutar la simulación del modelo en hardware, XSG se encarga de realizar la síntesis, ruteo y configuración del subsistema de procesado en la placa conectada, permitiendo que el flujo de entrada de datos fluya desde el espacio de trabajo de MATLAB a la placa. La información es procesada dentro del hardware y devuelta al entorno de trabajo de MATLAB.

La diferencia entre la simulación y la co-simulación está dada no por el resultado funcional (diferencias en el procesado realizado), sino por la simulación temporal (diferencias en los tiempos de respuesta del sistema), ya que ésta está dada por el posicionado y ruteo de los componentes lógicos y específicos en el dispositivo físico.

Una vez obtenidos los resultados del procesado, ya sea por simulación o co-simulación, se realiza el procesado de la imagen por software a total precisión, para luego convertir la salida a enteros de 8 bits sin signo (imágenes en escalas de grises con 256 niveles). La

Figura 1.7: Co-simulación hardware de un sistema de procesado.

diferencia entre el procesado por hardware y el procesado por software brinda información de la fiabilidad del diseño realizado. Además, ésta es utilizada para determinar la efectividad de la parametrización de los bloques desarrollados, teniendo en cuenta los posible errores de cálculo debido a la configuración de los diferentes bloques básicos. Este tópico se analiza de en cada bloque desarrollado.

Capítulo 2
Filtros de procesado lineal

SE clasifica en procesado lineal de imágenes a todo aquel algoritmo de procesamiento que utiliza una transformación lineal para lograr la imagen resultante. La convolución de dos dimensiones es la operación característica de este procesamiento. La técnica de convolución permite diversos resultados en las imágenes procesadas, entre ellas la detección de bordes, el realce y el suavizado de la imagen. La implementación de este tipo de algoritmo sobre hardware reconfigurable ha encontrado aplicación práctica en diferentes áreas, siendo una necesidad común la rapidez en la ejecución de los mismos [3, 6, 9, 10, 16, 20, 22, 28, 32].

Este capítulo aborda la operación de convolución como algoritmo de procesamiento lineal, analizando la base matemática, las estructuras de implementación hardware realizadas para la biblioteca XSGImgLib y la comparación con otras arquitecturas.

2.1. Fundamento matemático

Muchas de las etapas de un sistema de procesado lineal de imágenes se basan en la operación matemática de convolución de la imagen, representada por una matriz de dos dimensiones, con otra matriz (de 3×3 ó 5×5 píxeles en la mayoría de los casos) denominada núcleo o kernel de convolución [9].

Dada la imagen A de $m \times n$ píxeles, representada en la Ecuación 1.2, y la matriz de convolución C de $h \times h$ elementos y radio $rw = \frac{h-1}{2}$ (donde h es 3, 5, 7 ó 9), dada en la Ecuación 2.1, se define la convolución, A_c, entre la imagen A y el kernel C mediante la operación matemática mostrada en la Ecuación 2.2, que se traduce en el sumatorio de la multiplicación de todos los píxeles de la imagen con el correspondiente valor del coeficiente del kernel rotado 180°(C') [3]. Para la Ecuación 2.2, $\alpha = rw - x + 1$ y $\beta = rw - y + 1$.

Además, se excluyen los bordes de la imagen ($rw < x < m - rw$ y $rw < y < n - rw$) donde no existe la información necesaria para realizar el cálculo. Como resultado de esta operación se obtiene una imagen procesada en todos los puntos de la imagen original menos en los bordes de la misma.

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1h} \\ c_{21} & c_{22} & \cdots & c_{2h} \\ \vdots & \vdots & \ddots & \vdots \\ c_{h1} & c_{h2} & \cdots & c_{hh} \end{bmatrix}. \tag{2.1}$$

$$A_c(x - rw, y - rw) = \sum_{i=x-rw}^{x+rw} \sum_{j=y-rw}^{y+rw} A(i,j)C'(i + \alpha, j + \beta). \tag{2.2}$$

Los algoritmos de convolución se basan en la estructura del kernel que define las operaciones a realizar. Estos núcleos se clasifican de acuerdo al resultado final del procesamiento. La Figura 2.1 muestra los resultados del procesado con los núcleos de convolución más utilizados: detección de bordes (*Sobel*, *Edge*, *Prewitt*, *Laplace*), suavizado (*Blur*, *Smooth*, *Gaussian*) y los de realce (*Sharpen*) de la imagen. La estructura característica de cada uno de los núcleos de convolución se ilustra en el Anexo 2.

Los kernels de convolución pueden cumplir con la característica de ser simétricos con respecto a la diagonal principal ($c_{ij} = c_{ji}$, $i \neq j$). La utilización de éstos permite la reducción del número de multiplicadores de la operación de convolución de h^2 a $(h^2 - \frac{h(h-1)}{2})$ [30]. Entre los kernels que cumplen con esta propiedad están *Edge*, *Sobel X-Y*, *Prewitt X-Y*, *Laplace*, *Blur*, *Sharpen*, *Gaussian* y *Smooth*.

Al trabajar con imágenes en modelos de representación definidos como RGB o escala de grises es preciso que las imágenes resultantes mantengan la escala original. Para ello es necesaria la normalización de los coeficientes del kernel de convolución entre los valores 0 y 1. Una forma de realizar esta operación es dividir cada coeficiente por la suma de todos

(a) Imagen original. **(b)** Filtro de detección de bordes (*Edge*).

(c) Filtro de emborronamiento (*Blur*). **(d)** Filtro de realce (*Sharpen*).

Figura 2.1: Resultado de la aplicación de diferentes filtros de convolución.

los coeficientes del kernel dada por:

$$S = \sum_{i=1}^{h} \sum_{j=1}^{h} C(i,j). \tag{2.3}$$

El proceso de normalización no puede realizarse de esta manera cuando el resultado de la Ecuación 2.3 es el valor 0. Como alternativa en este caso se puede optar por no normalizar los coeficientes del kernel o por utilizar, para esta operación, el valor máximo entre la suma de todos los coeficientes positivos y la suma de todos los negativos. Los efectos en la normalización del kernel están ligados a la cantidad de información que se mostrará en la imagen resultante, obteniéndose imágenes saturadas (Figura 2.2a), donde todo valor mayor que el máximo se interpreta como blanco, o normalizadas (Figura 2.2b), donde la

salida cumple con la escala de la imagen de entrada.

(a) Saturada. **(b)** Normalizada.

Figura 2.2: Cantidad de información de la imagen resultante en la operación de convolución.

2.2. Arquitecturas hardware de procesamiento lineal de imágenes de la biblioteca XSGImgLib

XSGImgLib, como se menciona anteriormente, es el resultado de esta investigación. Los bloques de procesado presentados a continuación utilizan las especificaciones planteadas en la Subsección 1.3.3, tanto para el diseño de la unidad de procesado, descrita para cada uno de los bloques de creados, como para el desarrollo del modelo de simulación y co-simulación, para la comprobación del sistema.

La paleta de procesamiento lineal de la biblioteca XSGImgLib se compone por cuatro filtros de convolución genéricos para ventanas de 3×3 y 5×5, nueve filtros específicos para ventanas de 3×3 y tres filtros específicos para ventanas de 5×5. La Tabla 6.1 del Anexo 1 presenta las características de cada uno de los bloques de esta paleta.

2.2.1. Convolución genérica no simétrica

El algoritmo general de procesado lineal de imágenes es la convolución no simétrica mostrada en la Figura 2.3. Éste está basado simplemente en la definición matemática de la función de convolución de dos dimensiones, calculando cada píxel de la imagen como se

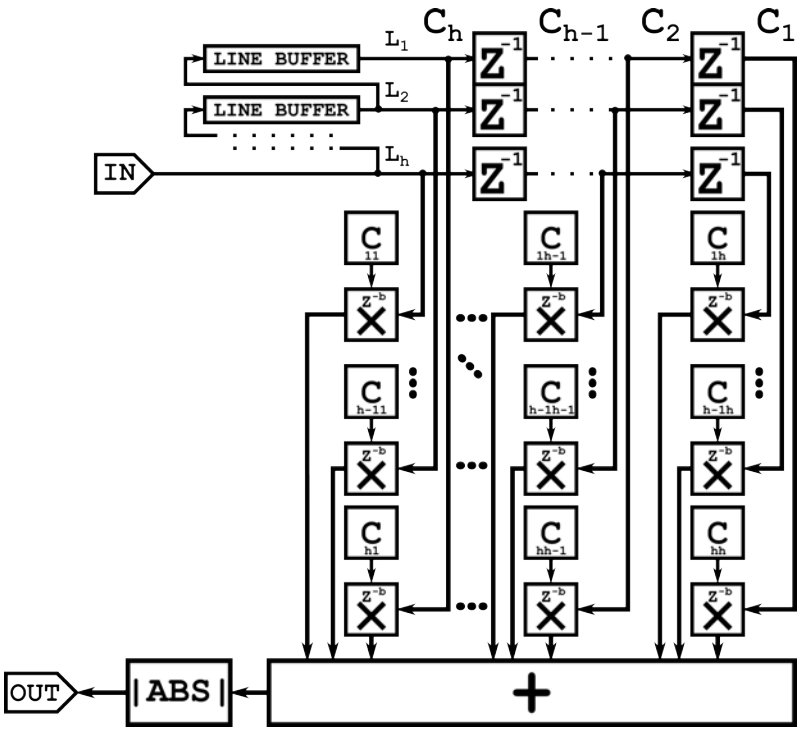


Figura 2.3: Convolución no simétrica.

define en la Sección 2.1. La señales L_1 a L_h son las filas de la imagen en la ventana de procesado, mientras que las señales C_1 a C_h representan las columnas en la misma.

Esta arquitectura es utilizada por dos bloques de convolución de la biblioteca, uno realizando cálculos a total precisión, mientras que otro permite la configuración de la cantidad de bits de la parte decimal del cálculo (el bloque normalizado, descrito en la Subsección 2.2.3, posibilita limitar la parte decimal de los coeficientes del kernel, así como los resultados de las operaciones de cálculo). Este mismo hecho se cumple para la arquitectura del bloque de convolución simétrica de la Subsección 2.2.2.

La estructura interna del diseño se compone de bloques de multiplicación (implementados utilizando los recursos dedicados del FPGA), de bloques de constantes, de bloques de suma/resta y de un bloque de cálculo del módulo o valor absoluto del resultado de la convolución (ver Anexo 3). Este último bloque es necesario ya que los valores de los píxeles de la imagen deben ser positivos. Además, el mismo ajusta la salida a 8 bits para la representación de la parte entera y 0 bits para la parte decimal, mediante un proceso de redondeo y saturación.

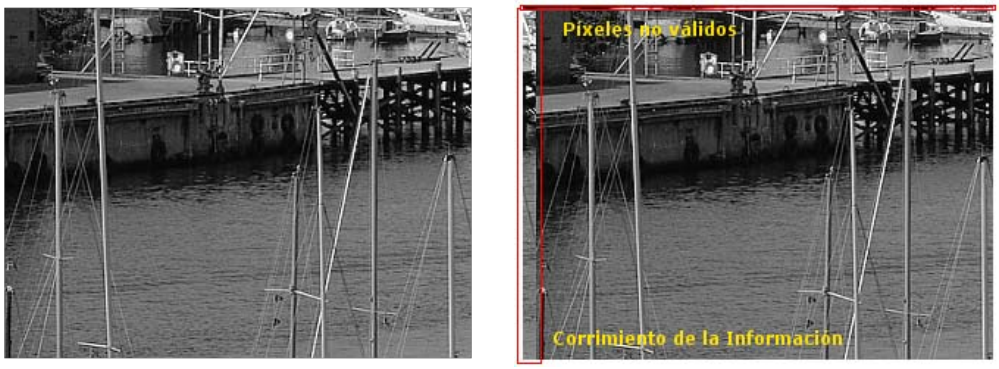
Los coeficientes del kernel, luego de ser seleccionados y normalizados en la configuración, son almacenados en los bloques de constantes, permitiendo ser cambiados con posterioridad en tiempo de diseño. La normalización del kernel se lleva a cabo utilizando los planteamientos de la Sección 2.1. En caso que el resultado de la Ecuación 2.3 sea 0, no se realiza la normalización del kernel, obteniendo una imagen resultante saturada por las operaciones del bloque del cálculo del valor absoluto.

El funcionamiento del sistema presenta una latencia (τ_s), dada por el tiempo requerido para completar los almacenadores de línea (τ_{lb}) y ocupar los registros (τ_{rg}), así como la demora interna de la operación a realizar (τ_{op}). Esta latencia se ejemplifica con la aparición de información no válida y el corrimiento de los píxeles en la imagen resultante, como muestra Figura 2.4. La imagen de la Figura 2.4b difiere de la procesada sin latencia (Figura 2.4a) en la aparición de píxeles sin información en la parte superior y en el corrimiento de los píxeles de la parte derecha a la parte izquierda de la imagen. Este efecto está siempre presente en los sistemas con retardo, ya sea por el proceso de paralelización o por las demoras en los bloques internos que forman la unidad de procesado. Un sistema basado en una convolución genérica presenta una latencia que se obtiene utilizando la Ecuación 2.4, donde la latencia de la operación (τ_{op}) para los diferentes bloques de procesado se describe en la Tabla 6.10. Los demás parámetros se definen en la Ecuación 4.13 y la Ecuación 4.14.

$$\tau_s = \tau_{lb} + \tau_{rg} + \tau_{op}. \tag{2.4}$$

Como se muestra en la Figura 2.3, luego del proceso de paralelización del flujo de datos de la imagen (utilizando los almacenadores de línea y los registros) se obtiene la información paralela necesaria para el cálculo del valor del píxel resultante. Después de este tiempo los h^2 píxeles se multiplican por los valores correspondientes del kernel rotado 180°, obteniendo el valor para la posición central de la ventana. En la próxima entrada de un bit serie, las estructuras de paralelización aseguran el movimiento de la ventana sobre la imagen, permitiendo el cálculo del valor para el píxel siguiente.

Los bloques de multiplicación (Figura 6.4) presentan dos entradas y una salida. La configuración de estos bloques permite la selección de la precisión del resultado de la operación, la latencia y la utilización de bloques dedicados o de bloques de lógica para su implementación, entre otros parámetros [37]. Éstos son abordados en la Sección 2.3.



(a) Imagen procesada sin latencia. **(b)** Imagen procesada con latencia.

Figura 2.4: Efectos de la latencia en el procesado.

TABLA 2.1: Consumos de recursos: convolución no simétrica de 5×5 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	818	25	0
	Porciento	13.89 %	125 %	0 %

Los bloques de suma/resta permiten seleccionar la operación (suma o resta) a realizar entre sus dos entradas, así como la latencia y la precisión del cálculo. Al presentar sólo dos entradas es necesario realizar operaciones de suma o resta en cascada para obtener el cómputo con más de dos valores. Mientras la cantidad de entradas en las operaciones en cascada sea impar se necesita el uso de bloques de retardos para mantener el sincronismo de las mismas.

La implementación hardware de este bloque requiere la utilización de gran cantidad de recursos del FPGA, entre ellos los bloques de multiplicación dedicados (h^2). La cantidad de multiplicadores del diseño imposibilita la utilización de un bloque de convolución de 5×5 en una placa *Spartan-3A Starter Kit 700a* como se muestra en la Tabla 2.1 [35].

Dado el consumo de los bloques de multiplicación, sumándole la necesidad del ahorro de recursos del FPGA, se desarrollan bloques de convolución parametrizables para minimizar estos efectos.

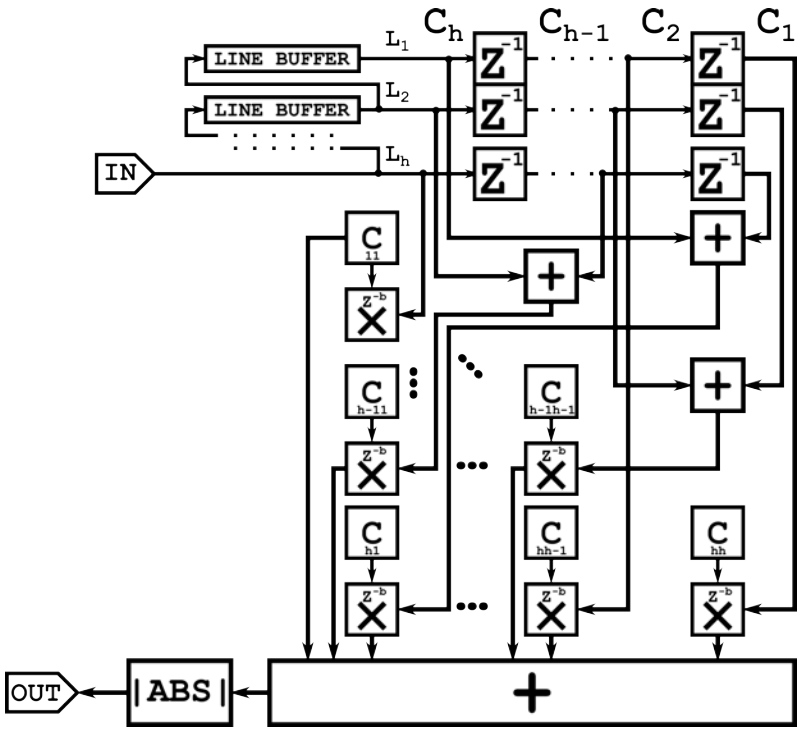


Figura 2.5: Convolución simétrica.

2.2.2. Convolución genérica simétrica

Como se menciona anteriormente, el uso de kernels simétricos permite el diseño de un bloque de convolución que sustituya $\frac{h(h-1)}{2}$ bloques de multiplicación por bloques de suma. El algoritmo de la operación es el mismo descrito en la sección anterior, pero utilizando la simetría del kernel de convolución.

La arquitectura, mostrada en la Figura 2.5, presenta la suma de las posiciones simétricas con respecto a la diagonal principal (posiciones L_aC_b y L_bC_a, donde $a \neq b$) antes de ser multiplicadas por uno de los valores del kernel. Los píxeles de la diagonal principal son multiplicados por su respectivo valor del coeficiente del kernel y luego se suman entre ellos evitando ciclos de espera innecesarios en las operaciones siguientes. Luego de las dos primeras etapas del algoritmo, éste continúa realizando sumas en cascada hasta obtener el valor resultante del píxel.

La reducción de los bloques de multiplicación permite la utilización de una placa de desarrollo con recursos limitados, como se muestra en la Tabla 2.2 para una *Spartan-3A Starter Kit 700a*.

TABLA 2.2: Consumos de recursos: convolución simétrica de 5×5 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	608	15	0
	Porciento	10.33 %	75 %	0 %

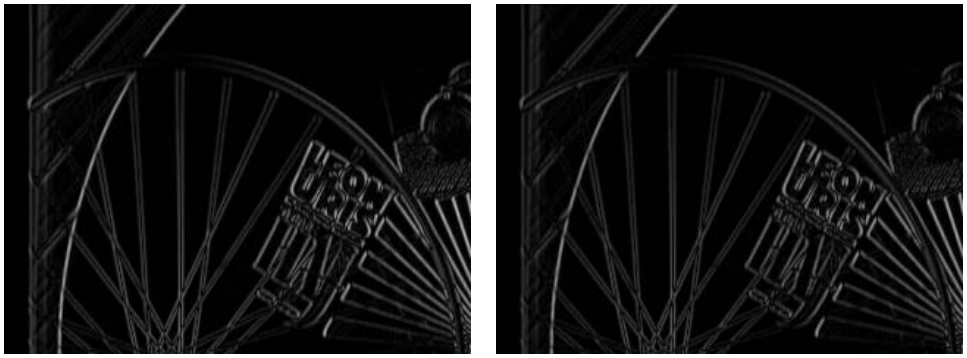
2.2.3. Convolución genérica normalizada

Los bloques básicos de XSG permiten configurar diversos parámetros los cuales se describen en el Anexo 3. Utilizando la parametrización de la salida de los operadores matemáticos así como los valores de las constantes es posible reducir el consumo de recursos en el dispositivo. Cómo se menciona en la Sección 2.1, al normalizar el kernel de convolución se hace necesario el trabajo con lógica de punto fijo, ya que los valores del kernel se ajustan entre 0 y 1. El resultado de las operaciones con éstos coeficientes generan valores con punto decimal, pudiendo limitar la representación de éste para lograr mejorar el consumo de recursos del diseño, a costa de errores en el cálculo. En los casos estudiados se puede llegar a un compendio entre la precisión del algoritmo y el consumo del mismo.

La arquitectura de estos bloques de procesado cumplen con las mismas características que las descritas en la Subsección 2.2.1 y la Subsección 2.2.2 para los bloques no simétricos y simétricos respectivamente. Su diferencia radica en la posibilidad de la elección de la cantidad de bits con que se representa la parte decimal de los coeficientes del kernel y las operaciones matemáticas. El Anexo 3 muestra las propiedades de cada uno de los bloques básicos utilizados en el diseño, entre ellas, la configuración de la salida.

La parametrización comienza con los bloques de constantes que almacenan los valores del kernel. La normalización del kernel de convolución implica la utilización de aritmética de punto fijo para los valores de sus coeficientes y los bloques de cálculo a trabajar con éstos. Los bloques de convolución normalizada utilizan la normalización planteada en la Sección 2.1. Cuando la Ecuación 2.3 tiene como valor 0 se realiza la normalización del kernel utilizando el valor máximo entre la suma de los valores positivos del kernel y el valor absoluto de la suma de los valores negativos del mismo.

Los bloques de constantes permiten la configuración de los bits para la representación de



(a) 3 bits y MSE de 9.8. **(b)** 8 bits y MSE de 0.029.

Figura 2.6: Imágenes normalizadas con diferente cantidad de bits para la representación de la parte decimal.

los valores almacenados, tanto de la parte entera como de la parte decimal. Este último parámetro es configurable por el usuario mientras que el total de bits para representar cada uno de los valores del kernel depende además de la cantidad de bits necesarios para la parte entera más un bit para el signo, si el coeficiente es negativo. Un valor que necesite mayor cantidad de bits para representar la parte decimal es truncado a la cantidad de bits seleccionada. Los demás bloques de cálculo del algoritmo truncan la salida para mantener una cantidad de bits fijos en cada operación. Al final la operación del cálculo del valor absoluto del píxel realiza la conversión a valores enteros de 8 bits para la representación de una imagen en escala de grises con 256 niveles.

La elección de este parámetro puede llevar asociada la ocurrencia de un error en el cálculo matemático del algoritmo, aunque la información pictórica recibida por el ojo humano no varía considerablemente. La Figura 2.6 muestra las diferencias del procesado con distinta precisión. Esta diferencia es cuantificable calculando el error cuadrático medio normalizado (MSE) de la resta entre la imagen procesada con toda la precisión de cálculo y la obtenida utilizando un bloque de precisión limitada. El MSE debe ser valorado por el usuario para decidir la configuración de este parámetro según las necesidades del diseño. Teniendo en cuenta que muchas de estas operaciones son aplicadas a imágenes que posteriormente van a ser segmentadas (donde se pierde parte de la información por la umbralización), este bloque puede ser útil al usuario a la hora de reducir la utilización de recursos.

2.2.4. Diseño de filtros específicos

La presencia de coeficientes que se repiten en un determinado kernel de convolución posibilita la optimización del consumo de recursos y la frecuencia de funcionamiento, diseñando algoritmos de convolución específicos para dicho kernel. La definición matemática de la función de convolución, expuesta en la Ecuación 2.2, es utilizada como base para estos diseños. Los filtros específicos permiten desarrollar arquitecturas óptimas en comparación con los filtros genéricos. La sustitución de los bloques de multiplicación por desplazamientos y sumas reduce asimismo la latencia de los diseños, ya que los bloques de cálculo son los causantes de ésta en la unidad de proceado. Cabe resaltar que la latencia debido a los bloques de paralelización se mantiene, ya que la misma es inherente al proceso de recepción de la imagen y a la necesidad del procesado en paralelo.

Estos bloques no son parametrizables pues se implementan remplazando los recursos dedicados de las versiones genéricas. La normalización del kernel sólo es realizada cuando la suma de los valores del kernel (Ecuación 2.3) es distinta de cero, en otro caso el resultado no es normalizado, obteniéndose una imagen saturada. Esto permite que el usuario final, conociendo la distribución de parámetros del kernel, realice la normalización de la salida como estime necesario. Dado que los diseños se obtienen para un filtro ya definido las operaciones intermedias de los algoritmos son optimizadas para minimizar la ocurrencia de errores en los cálculos intermedios, así como la cantidad de recursos lógicos.

Como se menciona en la Subsección 2.2.1 existe un proceso de paralelización de la información que influye en la latencia del sistema de procesado. Las arquitecturas de estos bloques coinciden en la estructura del algoritmo de paralelizado, difiriendo en el procesamiento. En los filtros específicos se integran los registros en el bloque de procesado, implicando que la demora del sistema se obtenga por la Ecuación 2.5 dónde la latencia de la operación se muestra en la Tabla 6.10 y la latencia de los almacenadores de línea (τ_{lb}) se define en la Ecuación 4.13.

$$\tau_s = \tau_{lb} + \tau_{op}. \tag{2.5}$$

Las arquitecturas que se presentan a continuación son versiones adaptadas de los diseños reales, siguiendo las pautas descritas en la Subsección 1.3.3 para el diseño de sistemas

TABLA 2.3: Consumo de recursos: filtro específico *Edge* optimizado en área o velocidad en una *Spartan-3A SK 700a*.

Recursos	Área		Velocidad	
	Utilizados	Porciento	Utilizados	Porciento
Slices	48	0.41 %	135	1.15 %
Mult 18×18	0	0 %	0	0 %
BRAM	0	0 %	0	0 %
Frecuencia	417.188 MHz		220.702 MHz	

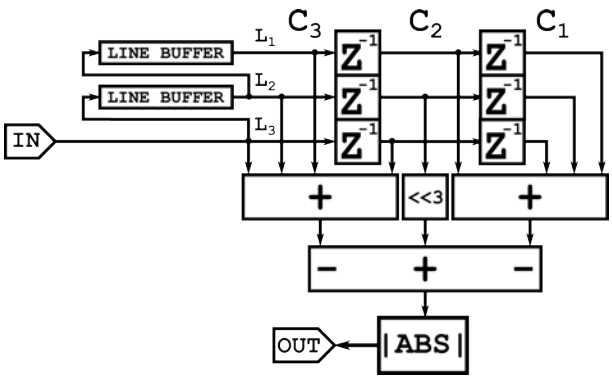
de procesado de imágenes. Esto permite mostrar la información necesaria de una manera simplificada. Los bloques de suma/resta con múltiples entradas (mayor que dos) se obtienen con varios bloques básicos brindados por XSG. La implementación práctica de los diseños demuestra la ineficacia del uso de demoras en los mismos para mejorar el pipeline de las operaciones y con ello la frecuencia de trabajo. Este efecto se ejemplifica en la Tabla 2.3, donde el uso de un sistema con pipeline (optimizado en velocidad) reduce la velocidad de ejecución en casi 1.9 veces y consume 2.8 veces más recursos.

2.2.4.1. Filtros específicos de detección de bordes

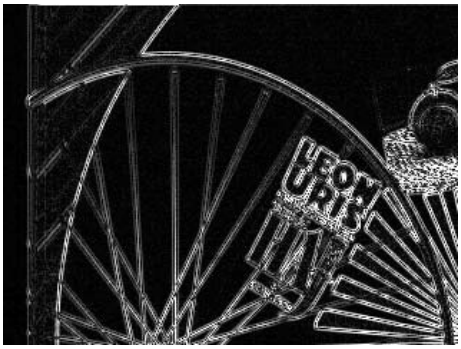
Entre los filtros de detección de bordes se encuentran los filtros *Edge*, *Laplace*, *Sobel* y *Prewitt*. Estos dos últimos se componen de tres filtros orientados en diferentes ángulos (eje X, eje Y y ejes X-Y). La aplicación de estos filtros obtiene como resultado una imagen en blanco, negro y algunas tonalidades de grises, donde sobresalen los bordes de la misma. Estos filtros no presentan normalización de los coeficientes puesto que la suma de los valores del kernel es cero. Las imágenes procesadas con los mismos presentan niveles de blanco saturados, necesitando la normalización de la salida por el usuario.

Arquitectura hardware del filtro específico *Edge*

Un filtro *Edge* puede ser implementado utilizando un bloque de convolución genérico simétrico, configurando los valores del kernel para esta operación. La estructura de un kernel *Edge* (ver Anexo 2) permite ser aprovechada para el diseño de un filtro específico. El píxel central, en la posición L_2C_2 de la Figura 2.7a, (multiplicado por el valor 8) es sustituido



(a) Arquitectura.



(b) Imagen resultante.

Figura 2.7: Filtro específico *Edge*.

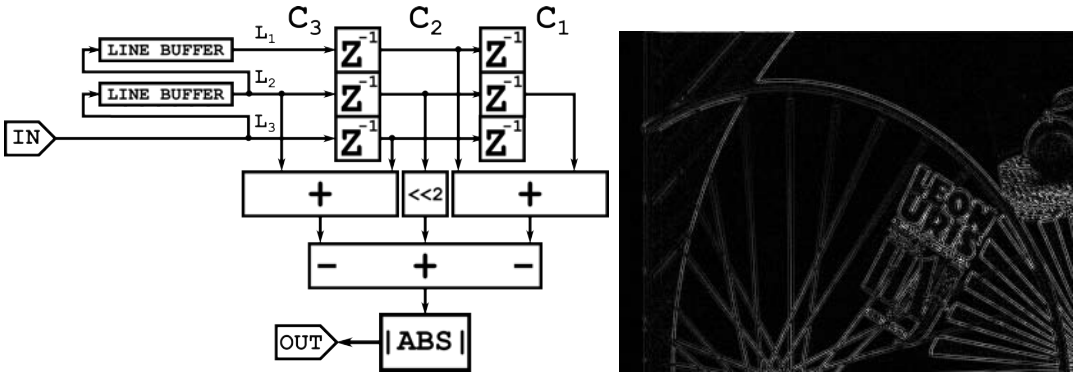
por una rotación de 3 bits a la izquierda, mientras que los demás píxeles (modificados por -1) son adicionados utilizando bloques de suma en cascada. Al resultado del cálculo del valor central se le resta la suma de los píxeles restantes para, por último, obtener el valor absoluto del píxel procesado.

El resultado de la aplicación de este filtro a una imagen en escala de grises que resalta los contornos de los objetos en la misma, como se aprecia en la Figura 2.7.

Arquitectura hardware del filtro específico *Laplace*

La estructura del kernel *Laplace* (ver Anexo 2) presenta ceros en las posiciones L_1C_1, L_1C_3, L_3C_3 y L_3C_1 de la Figura 2.8a. Esta característica evita el trabajo con las esquinas de la ventana a procesar, ya que el resultado de la multiplicación del píxel por su respectivo coeficiente del kernel es 0. El píxel central L_2C_2 (modificado por el valor 4) es rotado 2 bits a la izquierda mientras que los píxeles restantes, en las posiciones L_2C_1, L_1C_2, L_2C_3 y L_3C_2, (multiplicados por -1) son acumulados en paralelo. El resultado del píxel procesado se obtiene por el valor absoluto de resta del valor central modificado por 4 y la acumulación de los píxeles restantes, como se observa en la Figura 2.8a.

La Figura 2.8b muestra el resultado de la aplicación de un filtro con estas características, donde se obtiene una imagen con los bordes de los objetos en la misma delimitados. Este resultado presenta menor cantidad de color blanco, pues el nivel de grises de este filtro se ve menos afectado que el anterior por la no normalización de las operaciones.



(a) Arquitectura. **(b)** Imagen resultante.

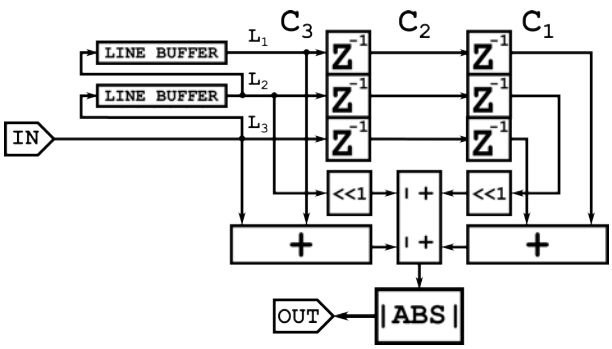
Figura 2.8: Filtro específico *Laplace*.

Arquitectura hardware del filtro específico *Sobel*

La arquitectura de los filtros *Sobel* y *Prewitt* dependen de la orientación deseada en el procesado. Un filtro *Sobel X* se compone de un kernel (ver Anexo 2) que presenta los valores en la primera y última columna (L_xC_3 y L_xC_1 en la Figura 2.9a, donde $1 \leq x \leq 3$), siendo simétrico a la columna central pero con diferentes signos. Los coeficientes de la columna central son cero por lo que los píxeles en esas posiciones no brindan información para el cálculo del resultado. Los valores centrales de las columnas primera y última (L_2C_3 y L_2C_1) se modifican por el valor 2, realizando una rotación de 1 bit a la izquierda del valor del píxel en estas posiciones. Los restantes valores de las columnas son sumados (modificados por 1 y -1 dependiendo de su posición) de acuerdo con los signos que presentan. El valor final del píxel resultante se obtiene luego de calcular el valor absoluto de la suma total de todos los cálculos intermedios teniendo en cuenta sus signos.

La estructura de un filtro *Sobel Y* (ver Anexo 2) presenta los mismos coeficientes que un *Sobel X*, pero organizados en filas en lugar de columnas. El diseño de la arquitectura de este filtro, mostrada en la Figura 2.10a, presenta una operación similar a la del filtro anterior.

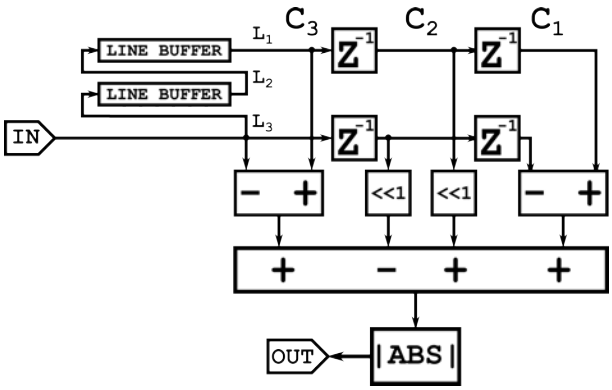
Un filtro *Sobel X-Y* se logra al aplicar un filtro *Sobel X* y un *Sobel Y* a la misma imagen y realizar la suma de los ambos resultados. Al trabajar con funciones lineales se puede obtener el kernel de convolución de un filtro *Sobel X-Y* con la suma de los kernel *Sobel X* y *Sobel Y*. Su estructura (ver Anexo 2) presenta valores ceros en la diagonal no principal (posiciones L_1C_3, L_2C_2 y L_3C_1 de la Figura 2.11a) y valores unos, positivos o negativos,



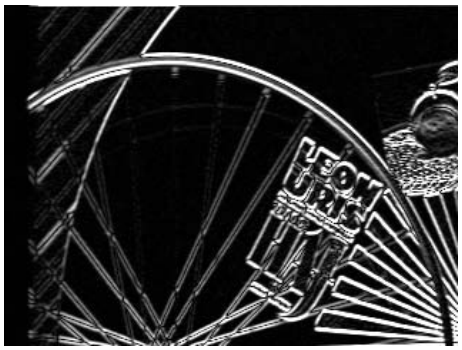
(a) Arquitectura.

(b) Imagen resultante.

Figura 2.9: Filtro específico *Sobel X*.



(a) Arquitectura.



(b) Imagen resultante.

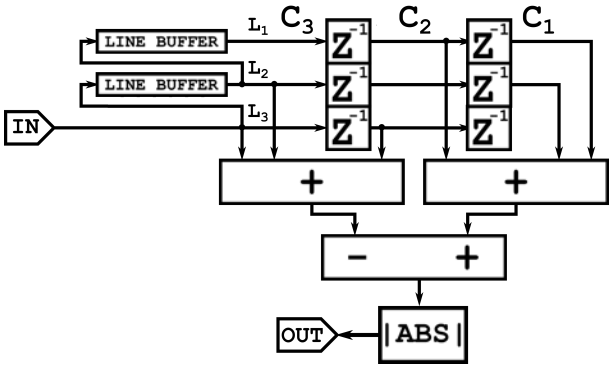
Figura 2.10: Filtro específico *Sobel Y*.

en el resto de los coeficientes. Los píxeles modificados por 1 ó -1 son sumados, teniendo en cuenta el signo, utilizando bloques de suma/resta en cascada. El píxel resultante se obtiene del cálculo del valor absoluto de la suma de las operaciones anteriores.

Las Figuras 2.9b, 2.10b, 2.11b ejemplifica la aplicación de los distintos filtros *Sobel*. Los resultados divergen en los ángulos de los bordes detectados, de acuerdo con la dirección del kernel.

Arquitectura hardware del filtro específico *Prewitt*

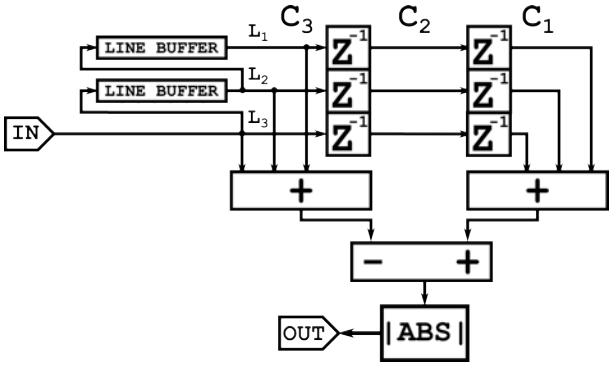
La arquitectura del filtro *Prewitt* presenta gran similitud a la del filtro *Sobel*. Su diferencia radica en la modificación de dos coeficientes en el kernel, cambiando los valores 2 por 1 de las versiones *X* e *Y* (posiciones L_2C_1 y L_2C_3 en las Figuras 2.12a y 2.13a) y los valores



(a) Arquitectura.

(b) Imagen resultante.

Figura 2.11: Filtro específico *Sobel X-Y*.



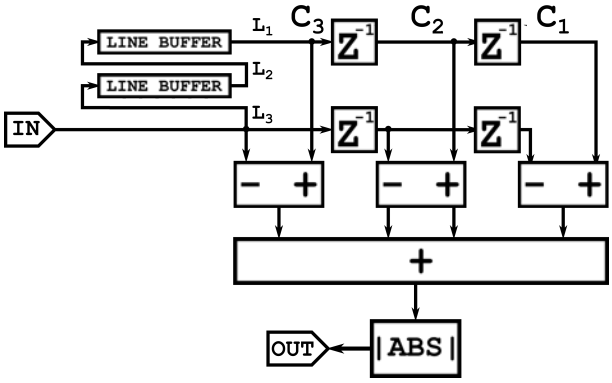
(a) Arquitectura.

(b) Imagen resultante.

Figura 2.12: Filtro específico *Prewitt X*.

de 1 de las esquinas de la diagonal principal por 2 en la versión *X-Y* (posiciones L_1C_1 y L_3C_3 en la Figura 2.14a). Ambos cambios mantienen los signos que presentaban antes del mismo. Las versiones *X* e *Y* presentan píxeles modificados sólo por valores 1 y -1 por lo que el píxel resultante se obtiene utilizando sumas y restas en cascada según los signos de los coeficientes. En cambio la versión *X-Y* presenta modificadores por 2, implementados utilizando rotaciones de 1 bit a la izquierda.

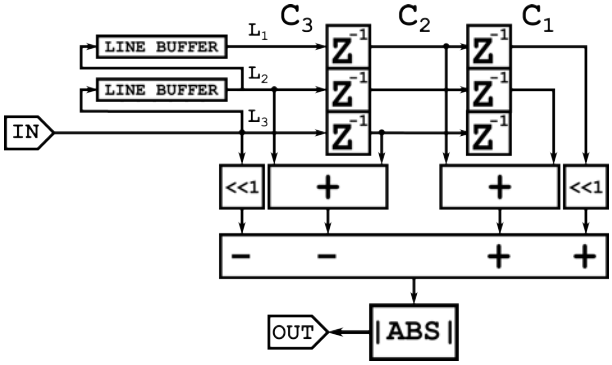
Las Figuras 2.12b, 2.13b, 2.14b muestran los resultados de la utilización de los diferentes filtros *Prewitt*, coincidiendo con las diferencias en los ángulos del los filtros *Sobel*.



(a) Arquitectura.

(b) Imagen resultante.

Figura 2.13: Filtro específico *Prewitt Y*.



(a) Arquitectura.

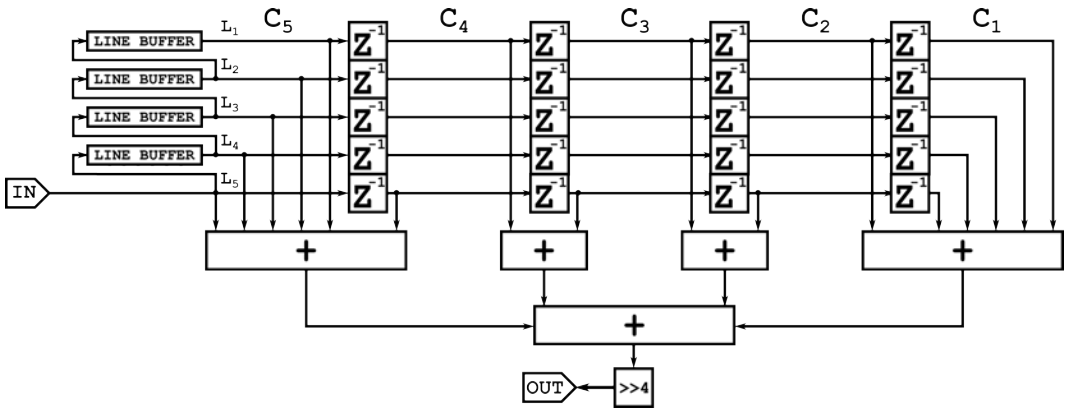
(b) Imagen resultante.

Figura 2.14: Filtro específico *Prewitt X-Y*.

2.2.4.2. Filtros específicos de suavizado

Los filtros de suavizado son aplicados a imágenes ruidosas o para desenfocar las mismas. El desenfoque es utilizado como un filtro de preprocesado para remover pequeños detalles antes de otra operación, rellenando pequeñas diferencias en líneas o curvas [9]. La aplicación de estos filtros modifica toda la información incluyendo los bordes de los objetos.

Las estructuras de los núcleos de suavizado (ver Anexo 2) no presentan valores negativos, posibilitando la normalización de la operación dentro del bloque de la biblioteca XSGImgLib.



(a) Arquitectura.

(b) Imagen resultante.

Figura 2.15: Filtro específico *Blur*.

Arquitectura hardware del filtro específico *Blur*

Un filtro *Blur* se compone por valores unitarios en los bordes de la ventana de 5×5 (L_1C_x, L_5C_x, L_xC_1 y L_xC_5 en la Figura 2.15a, donde $1 \leq x \leq 5$) y valores nulos en el resto de la misma. La suma de los coeficientes del kernel tiene como resultado 16, por lo que es posible normalizar el mismo. La arquitectura propuesta en la Figura 2.15a implementa la suma de todos los píxeles en las posiciones de los bordes de la ventana utilizando sumas en cascada. Por último, se realiza la normalización de la operación dividiendo por la suma de los elementos del kernel. Al ser esta suma potencia de dos, la división se implementa con la rotación del valor 4 bits a la derecha.

El resultado de la aplicación de este filtro se muestra en la Figura 2.15b. Su ejecución desenfoca la imagen de entrada eliminando el ruido en la misma.

Arquitectura hardware del filtro específico *Smooth*

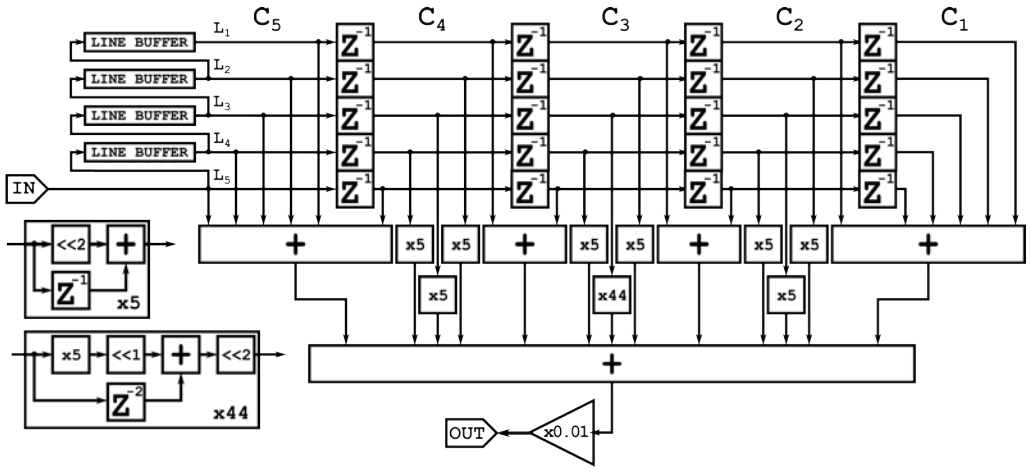
Los coeficientes de un filtro *Smooth* coinciden en la periferia con los de un filtro *Blur*. El píxel central de la ventana (L_3C_3 en la Figura 2.16a) está modificado por el valor 44 mientras que los píxeles restantes se multiplican por el valor 5. La arquitectura de este filtro requiere el uso de rotaciones y sumas para eliminar los bloques de multiplicación dedicados. La multiplicación por 5 se realiza rotando el valor dos veces a la izquierda (multiplicar por 4) y sumándole el valor original (bloque $\times 5$ en la Figura 2.16a). Para multiplicar por 44 se utiliza como base la multiplicación por 5 rotada una vez a la izquierda (multiplicar por 2) para obtener una multiplicación por 10. A ésta, a su vez, se le suma el valor original para multiplicar por 11, y por último se rota este valor dos veces a la izquierda (multiplicar por 4) para lograr el resultado esperado (bloque $\times 44$ en la Figura 2.16a).

Todos los valores de los píxeles, ya modificados son acumulados obteniendo un valor no normalizado. La suma total de los coeficientes del kernel es 100 por lo que es necesario dividir por este valor para normalizar resultado. Como la suma no es potencia de dos se multiplica por una constante con el inverso de su valor ($1/100$), introduciendo un pequeño error en el cálculo. La Figura 2.16b muestra el resultado de la aplicación de este filtro con un MSE de 0.013. Este filtro, como el anterior, realizar el desenfoque de la imagen pero afecta menos los bordes de los objetos.

Arquitectura hardware del filtro específico *Gaussian*

La estructura de un filtro *Gaussian* presenta modificadores en los píxeles por los valores 8, 4, 2, y 1. El píxel central L_3C_3 (multiplicado por 8) es rotado tres veces a la izquierda, los píxeles a cada lado del mismo (en las posiciones L_3C_2, L_2C_3, L_3C_4 y L_4C_3 de la Figura 2.17a) se rotan dos veces a la izquierda (multiplicar por 4), los píxeles modificados por el valor 2 son rotados una vez a la izquierda mientras que los restantes píxeles son sumados paralelamente. Luego del cálculo inicial, el valor del píxel saturado se obtiene por la suma en cascada de los resultados parciales.

La suma de los coeficientes del kernel es 52 por lo que es necesario multiplicar por una constante con el inverso de este valor ($1/52$) para obtener la imagen normalizada. La Figura 2.17b muestra el resultado de la aplicación de un filtro *Gaussian* con un MSE de 0.026.



(a) Arquitectura.

(b) Imagen resultante.

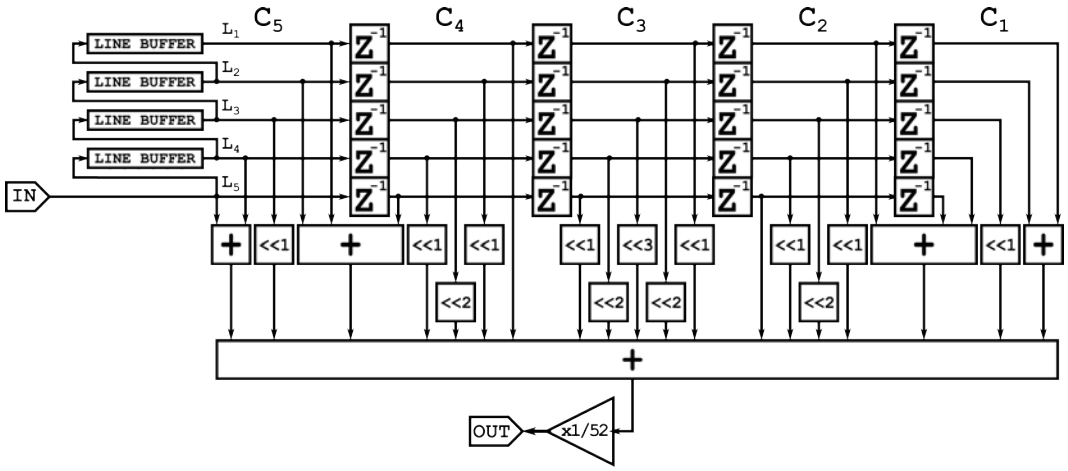
Figura 2.16: Filtro específico *Smooth*.

2.2.4.3. Filtro específico de realce

El principal objetivo de un filtro de realce es resaltar los detalles finos de la imagen o mejorar los fragmentos que fueron suavizados, tanto por error o por efectos naturales del proceso de adquisición. El realce está dado por una diferenciación espacial, desenfatizando las áreas con pequeñas variaciones en los niveles de grises y mejorando los bordes, además de otras discontinuidades, en la imagen [9].

Arquitectura hardware del filtro específico *Sharpen*

El kernel de un filtro *Sharpen* se compone por un elemento central con valor 32 y el resto con valor -2 (ver Anexo 2). Aplicando las propiedades de las transformaciones lineales, se



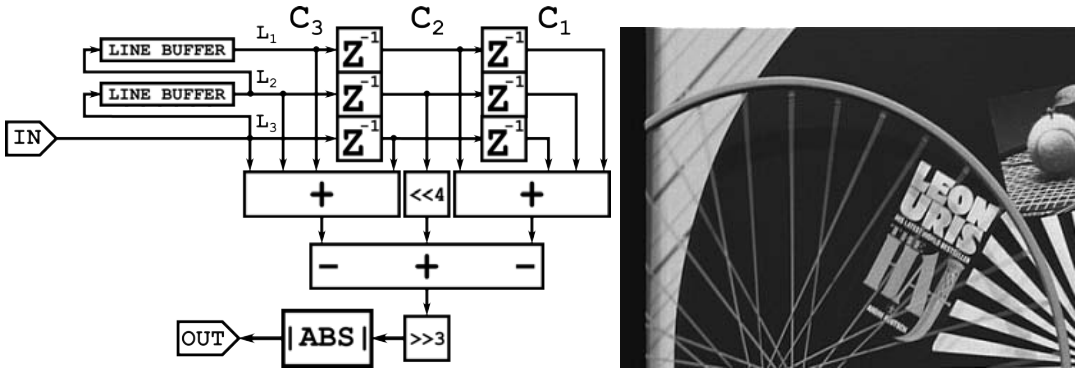
(a) Arquitectura.

(b) Imagen resultante.

Figura 2.17: Filtro específico *Gaussian*.

divide por 2 el núcleo, obteniendo 16 en el valor del coeficiente central, mientras que el resto de los valores se modifican por -1.

La arquitectura mostrada en la Figura 2.18a utiliza rotaciones de 4 bits a la izquierda para multiplicar el píxel central L_2C_2 por 16 y sumas en paralelo para los demás píxeles. Luego de esta primera etapa se continúan realizando sumas y restas en cascada hasta obtener el valor saturado del píxel procesado. La suma de los valores del kernel es 8, por lo que la normalización de la operación se realiza rotando 3 bits a la derecha el resultado, logrando una imagen como la mostrada en la Figura 2.18b, donde se resaltan los bordes de la misma.



(a) Arquitectura. **(b)** Imagen resultante.

Figura 2.18: Filtro específico *Sharpen*.

2.3. Parametrización de los bloques de convolución genéricos

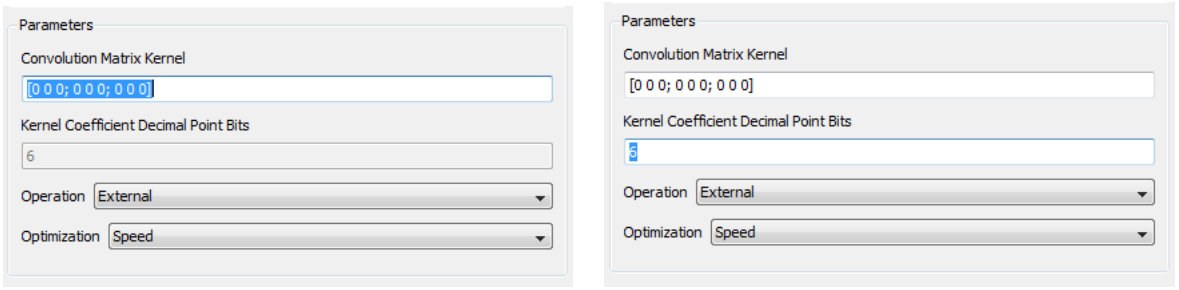
La parametrización de los algoritmos genéricos de convolución mejora el rendimiento y la utilización de recursos en comparación con las versiones no parametrizadas. Aunque nunca superan los valores alcanzados por los filtros específicos, planteados en la Subsección 2.2.4, la parametrización de los algoritmos genéricos es una solución al consumo de recursos, para los kernel no definidos con anterioridad.

Los parámetros configurables para estos bloques de procesamiento son la selección del kernel de convolución, la optimización de recursos de acuerdo a las necesidades del usuario, y la normalización de las operaciones de cálculo. La integración de varias de estas configuraciones posibilita la obtención de un bloque mejor parametrizado, y por ende, más eficiente. Las opciones son accesibles mediante la ventana de configuración del bloque seleccionado como se muestra en la Figura 2.19.

2.3.1. Selección del kernel de convolución

Los kernels de convolución son valores previamente definidos para las operaciones deseadas (*Edge*, *Prewitt*, *Sobel*, etc.). Estos kernels son fácilmente seleccionables por medio de una lista desplegable del bloque en cuestión (campo *"Operation"* en la Figura 2.19).

La selección del valor *"External"* en el campo *"Operation"* permite declarar un nuevo kernel



(a) Precisión total. **(b)** Normalizada.

Figura 2.19: Ventana de propiedades de los bloques de convolución genéricos.

TABLA 2.4: Operaciones para los bloques de convolución genéricos.

Convolución 3×3 **no simétrica**	**Convolución** 3×3 **simétrica**	**Convolución** 5×5 **no simétrica**	**Convolución** 5×5 **simétrica**
Sobel X	*Sobel X-Y*	Externo	*Blur*
Sobel Y	*Prewitt X-Y*		*Smooth*
Prewitt X	*Edge*		*Gaussian*
Prewitt Y	*Laplace*		Identidad
Externo	*Sharpen*		Externo
	Identidad		
	Externo		

de convolución no configurado, en la zona *"Convolution Kernel Matrix"*. Las operaciones configuradas para los bloques genéricos se definen en la Tabla 2.4.

La selección del kernel de convolución interviene en la operación a realizar por el bloque de procesado, pero no optimiza su ejecución, sólo son cambiados los valores de los bloques de constantes que almacenan los coeficientes del kernel. La existencia de cambios en el consumo de recursos está dada por la normalización de los coeficientes del kernel. Si un kernel es normalizado, la cantidad de bits para representar las operaciones de cálculo son ajustadas a las necesidades del mismo, por lo que da lugar a variaciones en el consumo de recursos utilizando la misma arquitectura. Estos parámetros se abordan en la Subsección 2.3.3.

2.3.2. Optimización en área o velocidad

Las operaciones de multiplicación, incluso utilizando bloques dedicados, requieren varios ciclos de reloj para completar el cálculo, por esto es necesario sincronizar los restantes elementos del circuito utilizando bloques de retardo, para optimizar la frecuencia de operación del sistema. El número de bloques de retardo, en combinación con los ciclos de demora de los multiplicadores, puede ser parametrizado de acuerdo a las necesidades de velocidad o consumo de recursos del diseño final.

Los bloques de multiplicación (ver Anexo 3) utilizan un parámetro para ajustar la cantidad de ciclos de reloj que demora el cálculo (*"Latency"*). Por defecto el mismo está parametrizado en el valor 3. Éste es el tiempo estimado que necesita el bloque para obtener el resultado. Si el parámetro es configurado con un valor menor que el óptimo el tiempo del ciclo de reloj del sistema aumenta para esperar a que el resultado de la multiplicación esté listo.

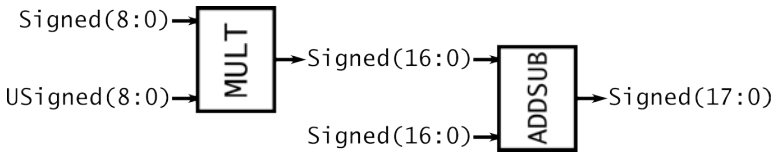
La optimización en área o velocidad, seleccionable en el campo *"Optimization"* de la ventana de configuración del bloque (Figura 2.19), utiliza este parámetro para obtener mejores resultados en las implementaciones. La optimización en área configura la latencia de los bloques de multiplicación con el valor 2 mientras que la optimización en velocidad lo hace con el valor 4. Estas latencias se obtienen de forma práctica configurando los bloques con diferentes valores, utilizando como referencia la latencia por defecto.

Los resultados de implementación del Anexo 6 (Tabla 6.11 a la Tabla 6.17) muestran las ventajas de esta parametrización. La utilización de recursos varia entre un 28.98 % y un 59.38 % menos para un kernel de 3×3, y entre 30.02 % y un 50.16 % para uno de 5×5. En cambio la optimización en velocidad permite un aumento de la frecuencia entre un 13.56 % y un 33.93 % para una ventana de 3×3, y entre 8.93 % y un 65.45 % para una de 5×5. Estos valores dependen tanto de la arquitectura del bloque a utilizar (simetría), como de la placa sobre la cual se implementa el sistema.

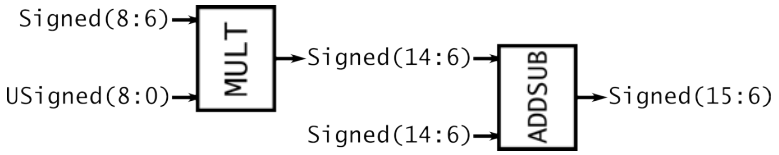
2.3.3. Normalización de las operaciones

Los coeficientes de los kernels de convolución son números enteros implicando que las operaciones con estos valores no requieran de lógica de punto fijo. Trabajando con imágenes en escala de grises y utilizando la arquitectura genérica, donde los valores se representan

como enteros de 8 bits sin signo (*usigned 8:0*), la multiplicación del valor de un píxel por un coeficiente no normalizado, representado por un valor entero con signo (*signed 8:0*), devuelve un número entero con signo representado con 16 bits (*signed 16:0*). Al normalizar el kernel de convolución los valores se representan con n bits entre los cuales se encuentran los bits para la parte decimal. Considerando 6 bits para representar la parte decimal de los coeficientes del kernel (*signed 8:6*), la multiplicación del valor de un píxel de la imagen por el coeficiente del kernel devuelve como resultado un valor fraccionario con signo representado en 14 bits de los cuales 6 se utilizan para la parte decimal (*signed 14:6*). La suma en cascada de los valores aumenta un bit en cada operación (Figura 2.20).



(a) Precisión total.



(b) Precisión normalizada.

Figura 2.20: Precisión de las operaciones.

Los bloques de convolución normalizada permiten la selección de la cantidad de bits para representar la parte decimal del bloque de convolución en el campo *"Kernel Coefficient Decimal Point Bits"*. La reducción de este valor influye en la cantidad de recursos a utilizar y en el error de cálculo por la limitación de la cantidad de bits para representar la parte decimal de los valores. Estas variaciones se ejemplifican en la Tabla 6.18 y la Tabla 6.19. En algunas ocasiones la reducción de la cantidad de bits de la parte decimal, conjunto con las especificaciones del kernel utilizado, eliminan la ocurrencia de un error (Tabla 6.19, Punto decimal de 3 bits). Ésto se debe a que las aproximaciones realizadas en la fase de normalización son contrarrestadas con las aproximaciones de los cálculos intermedios.

2.4. Comparación de arquitecturas

Las diferentes arquitecturas para procesado de imágenes basadas en convoluciones permiten la selección de las características necesarias para el desarrollo de aplicaciones específicas. Los bloques genéricos de convolución permiten la parametrización de los componentes internos de procesado, mientras que los filtros lineales específicos están optimizados en la ejecución de un algoritmo dado.

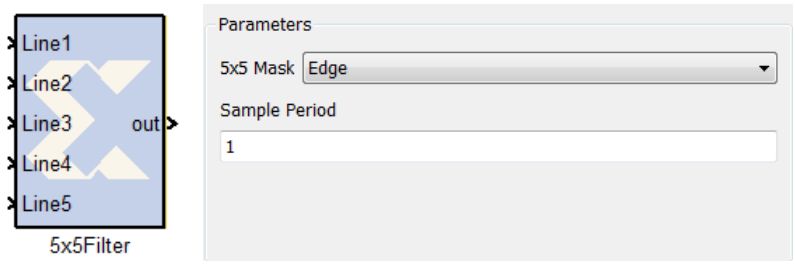
2.4.1. Convolución no simétrica y simétrica

La utilización de bloques de convolución simétricos es una ventaja ante los bloques de convolución no simétricos. La reducción de los multiplicadores permite que los diseños puedan ser implementados en placas con menos recursos específicos. La disminución en el consumo de recursos va desde un 18 % a un 22 % para la optimización en velocidad, mientras que en la optimización en área ronda el 11 %, como se puede observar en la Tabla 6.38, la Tabla 6.39 y la Tabla 6.40. Los bloques de multiplicación se reducen en un 33 % ó 40 % dependiendo de la arquitectura utilizada.

Como se menciona en la Subsección 2.2.1 y se muestra en la Tabla 2.1, la versión de 5×5 no simétrica no puede ser implementada en una placa de desarrollo *Spartan-3A SK 700a* porque ocurre un sobre consumo de los componentes de multiplicación. Siendo los bloques simétricos una solución para el consumo de recursos de las implementaciones, no pueden ser utilizados cuando los kernels de convolución no cumplen con estas características.

2.4.2. Convolución genérica y específica

Al conocer las operaciones a realizar sobre las imágenes es conveniente el uso de bloques específicos de filtrado. La Subsección 2.2.4 describe las arquitecturas de varios filtros específicos de procesamiento de imágenes. El consumo de recursos en comparación con las arquitecturas genéricas se muestra en el Anexo 7 (Tabla 6.41 a la Tabla 6.52). Por lo general el uso de arquitecturas genéricas disminuye el consumo de recursos desde un 17 % hasta un 60 % para los diseños optimizados en área, mientras que la frecuencia varía entre 1.33 y 3.20 veces más que en los diseños genéricos optimizados en velocidad. Un caso especial lo constituye el filtro específico Smooth el cual presenta un consumo de recursos mayor en



(a) Diagrama. **(b)** Propiedades.

Figura 2.21: Bloque de convolución genérico de 5×5 de XSG [37].

un 5 % sobre la versión genérica, dada por la complejidad de sus operaciones. En todos los casos se elimina la utilización de bloques de multiplicación dedicados.

Los filtros específicos tienen el inconveniente de la necesidad de remover el bloque e intercambiarlo por otro al querer realizar una nueva operación sobre la imagen de entrada. La utilización de bloques genéricos resuelve este inconveniente, posibilitando el cambio del filtro de procesado en tiempo de ejecución, mediante bloques de memorias accesibles por con un procesador embebido.

2.4.3. Convolución de la biblioteca XSGImgLib y de System Generator

Como se menciona en la Subsección 1.3.2 System Generator posee un bloque genérico de convolución de 5×5 con cinco entradas, una para cada fila de la ventana de procesado, y una salida (Figura 2.21a), cuyo consumo se muestra en la Tabla 6.20 y la Tabla 6.21 para diferentes placas de desarrollo. Este bloque permite la selección del kernel de convolución, mediante una lista desplegable, y el período de muestreo (Figura 2.21b).

Optando por los casos más críticos de la biblioteca XSGImgLib (la convolución genérica no simétrica de 5×5 y el filtro específico *Smooth*) el consumo de recursos lógicos logra un ahorro del 10 % y el 16 % para el filtro genérico optimizado en área y el filtro *Smooth* respectivamente. Para la optimización en velocidad la utilización de los recursos es un 42.52 % mayor que en el bloque de XSG.

El bloque de convolución de XSG consume cinco veces menos multiplicadores que la convolución genérica no simétrica. El número de multiplicadores se reduce al compararlo con

el bloque genérico simétrico y los bloques específicos, llegando a no necesitarlos en este último (Tabla 6.53). Al almacenar los valores de los coeficientes del kernel en bloques de memoria dedicada y al utilizar una arquitectura basada en bloques MAC, el diseño del componente de convolución de XSG utiliza 5 bloques de memoria (BRAM) mientras que los bloques genéricos de convolución y los filtros específicos de la biblioteca XSGImgLib no necesitan de estos recursos.

El uso de bloques MAC, con 5 unidades de procesado, establece una arquitectura paralela con 5 píxeles de entrada en un mismo ciclo de reloj. Para realizar el cálculo de la convolución de una imagen con una ventana de 5×5 se necesitan a la vez 25 píxeles, por lo que la arquitectura MAC requiere de 5 ciclos de reloj para que un píxel de salida esté listo. La frecuencia de operación del sistema debe ser 5 veces mayor, o el período de muestreo del bloque 5 veces menor, que un sistema completamente paralelo. Ésto implica que la frecuencia de operación del bloque sea mucho más baja que en las arquitecturas en paralelo, llegando a ser 5 veces menor que las arquitecturas genéricas y hasta 7 veces menor que la arquitectura del filtro específico *Smooth*, no siendo éste el más rápido de la biblioteca XSGImgLib.

Capítulo 3
Filtros de procesado no lineal

EL filtrado espacial no lineal permite remplazar el píxel a procesar por el resultado de una operación no lineal. La operación característica de este tipo de procesado es el ordenamiento, aunque otros tipos de procesamientos (exponencial y logarítmico) se incluyen en esta categoría. Los filtros de ordenamiento, y en particular el de mediana, permite eliminar el ruido impulsivo y de alta frecuencia (*salt & pepper noise*[1]), en las imágenes sin afectar los bordes de las mismas [5, 8, 9, 11, 22]. Este procesado tiene una amplia aplicación en diferentes sectores entre ellos la medicina para eliminar el ruido en las imágenes radiográficas [18].

El capítulo presenta los filtros de ordenamiento como parte del procesamiento no lineal, y el filtro de mediana como caso especial de éste, analizando el funcionamiento de los mismos y las arquitecturas hardware para su implementación. Además se aborda el bloque de umbralización como parte de la paleta de procesado no lineal de la biblioteca.

3.1. Principio de operación

El procesamiento espacial de ordenamiento (*Ranking*) se basa en la sustitución de los píxeles de la imagen por el valor ubicado en la posición requerida, en función del lugar que ocupan los píxeles ordenados en la ventana de procesado [8, 11, 18]. El filtro de mediana es un caso específico de este filtro de ordenamiento donde el valor de salida está en la posición central de esta ventana.

La Figura 3.1 muestra la operación de un filtro de mediana de 3×3. Luego del paso de

[1] Ruido sal y pimienta.

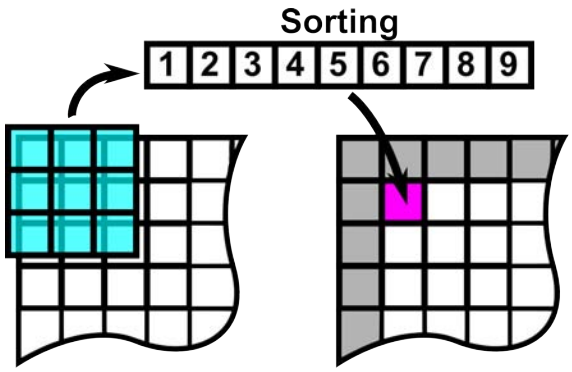


Figura 3.1: Operación de un filtro de mediana de 3×3.

la ventana sobre la imagen, los píxeles en ésta son ordenados obteniendo como salida el ubicado en la posición central.

Los filtros de ordenamiento permiten la selección de la posición del píxel de salida, posibilitando la obtención del valor mínimo y máximo de los mismos. Estas operaciones permiten implementar operadores morfológicos para las imágenes a procesar. Una imagen con ruido sal y pimienta (Figura 3.2a) al ser procesada con un filtro de ordenamiento para la mínima posición, resalta las tonalidades bajas de la imagen, aumentando los puntos negros en la misma (Figura 3.2b). La selección de la posición media suprime el ruido de la imagen evitando los cambios bruscos en los niveles de grises de la imagen (Figura 3.2c). En cambio, la posición máxima destaca las tonalidades altas, amplificando los punto blancos de la imagen (Figura 3.2d).

El aumento del tamaño de la ventana incluye un mayor número de píxeles en el proceso de ordenación, influyendo en la cantidad de ruido a procesar. Si la cantidad de ruido en la imagen es grande las ventanas de gran tamaño son ideales para contrarrestar el mismo, en cambio, si el nivel de ruido no es alto la imagen resultante es afectada por el suavizado de los contornos de los objetos en la misma. La región marcada en la imagen de la Figura 3.3b presenta un mayor desenfoque en los bordes de los objetos, mientras que este efecto se reduce al seleccionar un menor tamaño de la ventana (Figura 3.3a).

Otra operación no lineal es la umbralización utilizada en procesos de segmentación de imágenes. Muchas técnicas de procesado para sistemas empotrados se basan en la detección de contornos para delimitar regiones. El almacenamiento de estos datos puede ser redundante en ciertos diseños, en los que sólo se necesita conocer la presencia o no de contornos. La operación que realiza este procesado es llamada umbralización.

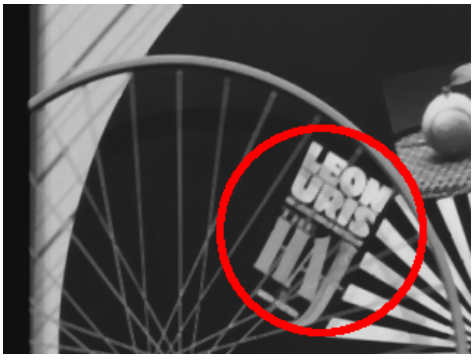
(a) Imagen original con ruido *salt & pepper*.

(b) Posición mínima.

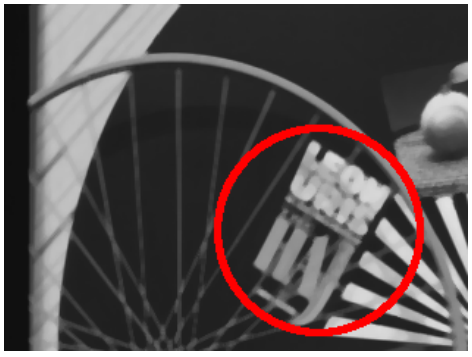
(c) Posición media.

(d) Posición máxima.

Figura 3.2: Resultado del procesado de imágenes con filtros de ordenamiento.



(a) Ventana de 3×3.



(b) Ventana de 5×5.

Figura 3.3: Diferencias de la eliminación de ruidos por filtros de ordenación con diferente ventana.

(a) Contornos de la imagen. **(b)** Imagen umbralizada.

Figura 3.4: Resultado del proceso de umbralización de imágenes.

El proceso de umbralización de una imagen consiste en analizar la información de la misma habilitando en la salida un valor "1" si el nivel de gris es mayor o igual al nivel deseado (T) como nivel máximo, y un valor "0" en caso contrario, como se define en la Ecuación 3.1 [11]. La elección del valor de umbralización está acompañado de la cantidad de información resultante. La Figura 3.4 muestra el resultado del proceso de umbralización de una imagen, luego de la aplicación de un filtro de detección de bordes a la misma. En ella se aprecia como sólo los contornos que cumplen con la condición del umbral son visibles en la imagen resultante.

$$A_s(x,y) = \begin{cases} 1, & si\, A(x,y) \geq T \\ 0, & si\, A(x,y) < T \end{cases}. \tag{3.1}$$

3.2. Arquitecturas hardware de procesamiento no lineal de imágenes de la biblioteca XSGImgLib

El filtrado espacial no lineal permite, mediante el filtro de mediana, solucionar defectos en imágenes que dificultan o confunden, tanto al observador en el momento del análisis visual como a los sistemas de procesado computarizados en la extracción de características de las mismas.

3.2.1. Filtro genérico de ordenamiento

El diseño del filtro de ordenamiento genérico de la biblioteca XSGImgLib se basa en la arquitectura no recursiva definida por Chakrabarti en [8]. El funcionamiento del mismo es una extensión del algoritmo de ordenación de una dimensión para un filtro de mediana no recursivo.

En un filtro de mediana unidimensional, una ventana de $2N+1$ muestras se desplaza por el arreglo de puntos devolviendo el valor correspondiente a la posición central en el arreglo ya ordenado (mediana). Sea W_i una ventana ordenada de una dimensión, como muestra la Ecuación 3.2, y_i es el valor de la posición media de W_i definida como $y_i = median\{W_i\}$. Para un filtro en dos dimensiones, una ventana de $k \times k$ elementos ($k = 2N+1$) se mueve sobre la imagen. Definiendo $W_{i,j}$ como una ventana en dos dimensiones, centrada en la posición (i, j) entonces $y_{i,j} = median\{W_{i,j}\}$ [8].

$$W_i = \{x_{i-N}, \ldots, x_i, \ldots, x_{i+N}\}. \tag{3.2}$$

Las arquitecturas para filtros de ordenamiento pueden clasificarse a groso modo en dos clases: una en la que los valores son almacenados en orden de llegada, y otra en la que las muestras se almacenan de forma ordenada [8]. La arquitectura planteada por Chakrabarti corresponde a la primera categoría.

Chakrabarti propone el uso de un arreglo compuesto por k^2 procesadores, donde el período de muestreo es función del tiempo de actualización de la posición de los elementos antiguos en la ventana, utilizando los resultados de las comparaciones con las nuevas muestras y con las muestras desechadas. Los procesadores del P_0 al P_{k-1} (Figura 3.6) almacenan k elementos de la nueva columna, procedente de los almacenadores de línea. Los procesadores del P_k al P_{k^2-1} (Figura 3.5) contienen las muestras antiguas en la ventana. Cada vez que se introduce una nueva columna a la ventana de procesamiento los k valores más antiguos son desechados por la última columna de procesadores [8].

La Figura 3.5 muestra la arquitectura de los procesadores del P_k al P_{k^2-1}. Éstos actualizan su posición (R') utilizando la comparación con los k nuevos elementos en la ventana (X_{NEW}) y los k elementos desechados de la última columna de procesadores. Cada uno de ellos presenta un registro de datos A (*Register A*) para almacenar el valor de la muestra

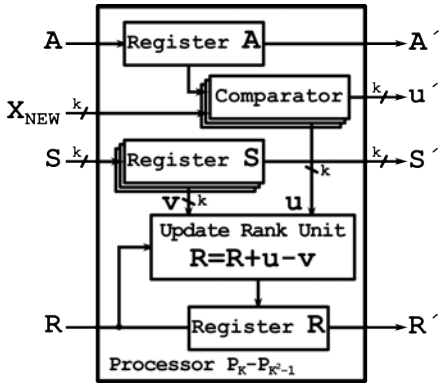


Figura 3.5: Procesadores P_k-P_{k^2-1}.

en la posición de la ventana, un registro R (*Register R*) que contiene la posición de la muestra ordenada con los elementos de la ventana, y k registros S (*Register S*) que almacenan los resultados de las comparaciones de la muestra con los restantes píxeles en la ventana [8]. Además, presentan k unidades de comparación (*Comparator*) y una unidad de actualización de la posición (*Update Rank Unit*). La presencia de los registros S evita realizar comparaciones con todas las muestras en la ventana, durante la actualización de las posiciones de los píxeles. Los valores de cada registro S se obtienen mediante las operaciones de los procesadores P_0 - P_{k-1}. Estos registros disminuyen su tamaño en un bit al ser desplazados de un procesador a otro.

La salida de cada unidad de comparación (k unidades por procesador) es un bit u que contiene el resultado de la operación entre las nuevas muestras y el correspondiente valor almacenado en el registro A, conformando un total de k bits u por procesador. De los registros S, de $k \times Columna\,del\,procesador - 1$[2], se obtiene el bit menos significativo (LSB) de cada registro (bit v) para la actualización de la posición de la muestra en el procesador.

Los procesadores del P_0 al P_{k-1} utilizan la arquitectura presentada en la Figura 3.6. La función de los mismos es calcular las posiciones de las k nuevas muestras (R'), contando los valores en la ventana que son menores que el píxel del procesador en cuestión. Este bloque se compone de una unidad de obtención de la posición (*Rank Calculation*), que opera con los resultados de las comparaciones de la muestra con los restantes valores en la ventana [8]. Los resultados de las comparaciones se reflejan en los valores de los $k(k-1)$

[2]El registro S se compone de k registros de tamaño igual al valor de la columna posicionada menos uno, contando como la columna k la de los procesadores P_0 - P_{k-1}.

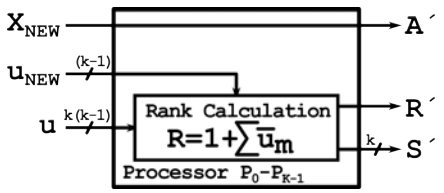


Figura 3.6: Procesadores P_0-P_{k-1}.

bits u y los $k-1$ bits u_{NEW}, que son entradas de la unidad de obtención de la posición.

Además de la posición de la muestra en el procesador, la unidad de obtención de la posición conforma los k registros S (S') para cada uno de los procesadores P_0 - P_{k-1}. La composición de este registro se realiza concatenando los bits u de la misma fila de procesadores, almacenando así el resultado de la comparación del valor de la muestra con los valores antiguos. Los valores de los bits u_{NEW} manifiestan los resultados de las comparaciones entre las k nuevas muestras. El bloque de comparación se diseña fuera del procesador para eliminar comparaciones redundantes en las unidades de procesado.

El diseño mostrado en la Figura 3.7 utiliza la arquitectura de Chakrabarti para el filtro de ordenamiento de dos dimensiones. Los valores de los k nuevos píxeles (X_{NEW}), procedentes de los almacenadores de línea y la posición deseada ($RANK$) son las entradas al bloque de procesado.

El funcionamiento del bloque comienza con la comparación, por todos los procesadores excepto los procesadores P_0 - P_{k-1}, entre el valor de la muestra almacenada en el registro A y los k nuevos píxeles (X_{NEW}). Estos resultados corresponden a los k valores de los bits u. Además, los LSB de los k registros S en cada procesador, los cuales contienen los resultados de las comparaciones entre la muestra del procesador y los k valores recientemente desechados, son asignados a los k valores de los bits v. El siguiente paso del algoritmo requiere la actualización del valor de la posición por estos procesadores. Para ello se hace uso de la posición anterior en la ventana, los resultados de las nuevas comparaciones y la actualización con respecto a las muestras desechadas, como se expone en la Ecuación 3.3 [8].

$$R = R + \sum_{m=0}^{k-1} u_m - \sum_{m=0}^{k-1} v_m. \tag{3.3}$$

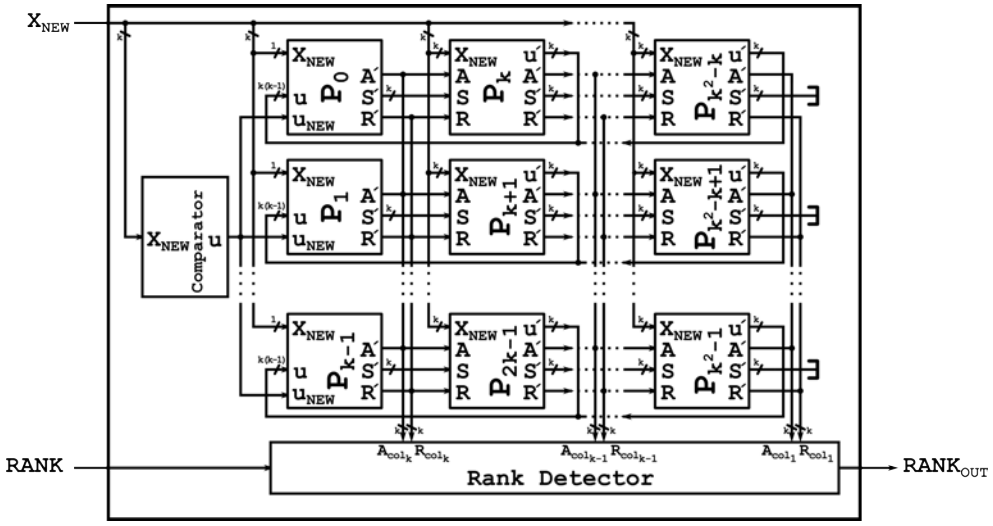


Figura 3.7: Arquitectura del filtro de ordenamiento genérico.

Paralelamente los procesadores del P_0 al P_{k-1} calculan la posición de las nuevas muestras. Esta operación utiliza la salida del comparador de muestras (*Comparator* en la Figura 3.7) que aporta los valores de u_{NEW} y los $k(k-1)$ bits del resultado de las comparaciones realizadas por los procesadores P_k - P_{k^2-1} con el correspondiente valor X_{NEW}. Este cálculo se muestra en la Ecuación 3.4, donde no se incluye la comparación de la muestra con ella misma ($i \neq m$). Además se obtienen los valores de los k registros S por procesador, concatenando los bits u de cada fila en la posición correspondiente a la columna. El LSB de este registro debe contener la comparación con el valor más antiguo de la fila en la ventana.

$$R = 1 + \sum_{i=0}^{k^2-1} u_m(P_i). \tag{3.4}$$

Por último el algoritmo procede a obtener la posición deseada. El detector de posición de la Figura 3.8 se compone de bloques de comparación y de multiplexado. Una vez obtenidas las posiciones de los píxeles en la ventana se realiza su comparación con la posición deseada. La concatenación de la salida de los comparadores habilita la entrada respectiva del multiplexor devolviendo el valor de la muestra correspondiente con la posición deseada al sistema de procesado.

El bloque de ordenamiento genérico no presenta demora en la ejecución, obteniendo una

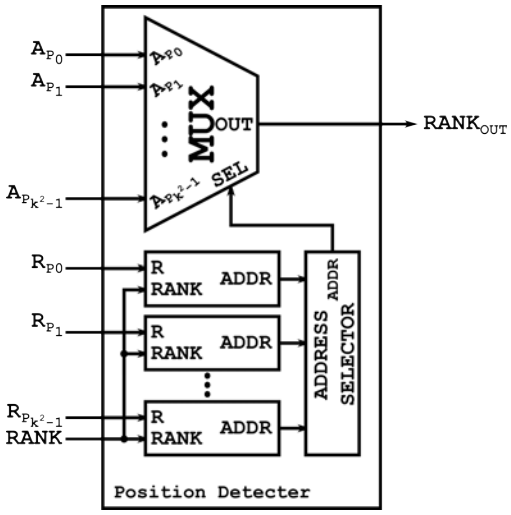


Figura 3.8: Detector de posición.

salida válida en cada ciclo de reloj. La latencia del sistema puede ser calculada mediante la Ecuación 2.5 utilizada para un filtro específico, donde la latencia de la operación (τ_{op}) se define en la Tabla 6.10 del Anexo 5. El Anexo 6 describe el consumo de recursos de la implementación del bloque en diferentes placas de desarrollo (Tabla 6.22 a la Tabla 6.25).

3.2.2. Filtro específico de mediana

El ahorro de recursos y la necesidad de aumento de velocidad para aplicaciones en tiempo real, como se ha planteado en capítulos anteriores, impulsan el desarrollo de nuevas arquitecturas más eficientes. La arquitectura específica de un filtro de mediana 2D para una ventana de 3×3 corrobora esta afirmación.

Un filtro de mediana puede ser implementado utilizando como base un bloque de comparación de dos elementos. Esta estructura permite el diseño de un filtro específico de cualquier tamaño de ventana, a expensas de una gran cantidad de comparaciones [15, 18]. El diseño planteado por Golston, utilizado en varias implementaciones hardware para procesado de imágenes, elimina las limitaciones del diseño anterior, permitiendo el desarrollo de un filtro de mediana de 3×3 optimizado [11, 32]. Ambos diseños pertenecen a la segunda clasificación de los algoritmos de ordenamiento mencionada en la subsección anterior.

La unidad principal de la arquitectura de procesado de Golston se basa en un módulo de ordenamiento de tres entradas (A, B y C) que retorna los píxeles en las posiciones que

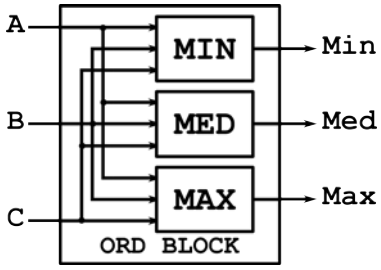


Figura 3.9: Módulo de ordenamiento de tres píxeles.

ocupan entre ellos. La arquitectura de este módulo, mostrada en la Figura 3.9, se compone de tres comparadores y tres multiplexores, habilitando, en la salida de estos últimos, los valores ordenados (Min, Med y Max).

Utilizando el bloque anteriormente descrito puede ser diseñado un filtro de mediana eficiente como se muestra en la Figura 3.10. Inicialmente los píxeles, provenientes de los almacenadores de línea ($L_1 - L_3$), son ordenados según sus valores. Cada columna se mantienen en el bloque de procesado durante tres ciclos de reloj, utilizando bloques de registros (Z^{-1}), hasta que se desecha por el movimiento de la ventana sobre la imagen. Los valores mínimos de las tres columnas ya ordenadas ($Min_1 - Min_3$) son introducidos a otro módulo de ordenamiento para la obtención del valor máximo entre los mínimos. Paralelamente se procesan los tres valores medios ($Med_1 - Med_3$) y los tres valores máximos ($Max_1 - Max_3$) mediante la misma unidad de ordenamiento, extrayendo los tres píxeles más cercanos a la posición central. Por último, estos tres resultados (Min_4, Med_4 y Max_4) son ordenados obteniendo el valor en la posición media [11, 32].

Esta arquitectura sólo puede ser implementada para una ventana de 3×3, por lo que para ventanas de mayor tamaño es necesario el uso de la arquitectura genérica. Luego del llenado de las columnas del sistemas, éste ofrece una salida en cada ciclo de reloj, presentando una latencia debida a la utilización de almacenadores de línea. Este tiempo es calculable por la Ecuación 2.5 definiendo el parámetro τ_{op} en el Anexo 5. La Tabla 6.26 y la Tabla 6.27 del Anexo 6 muestran la utilización de recursos del bloque para diferentes placas de desarrollo.

3.2.3. Bloque de umbralización

El bloque de umbralización presenta una arquitectura sencilla, mostrada en la Figura 3.11. Su composición cuenta con un módulo de comparación ($A>THR$) y un multiplexor (MUX).

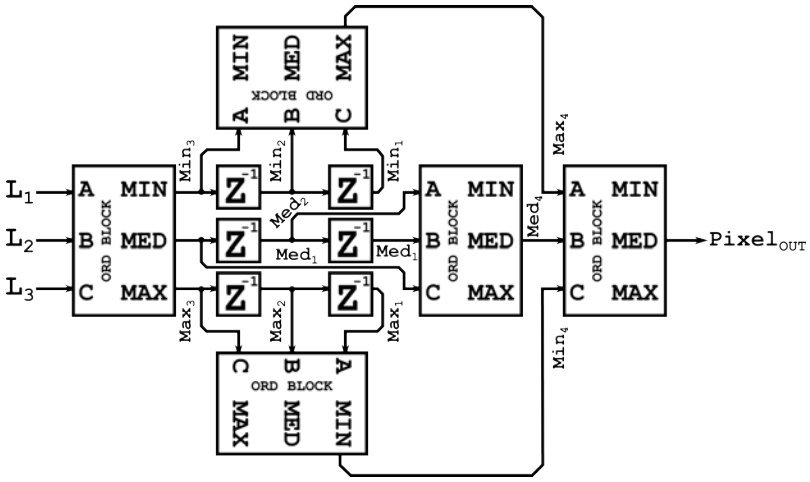


Figura 3.10: Arquitectura del filtro de mediana 2D de 3×3.

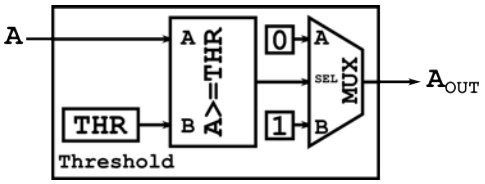


Figura 3.11: Arquitectura del bloque de umbralización.

Los píxeles de entrada al bloque son comparados con el valor de umbral deseado (*THR*), configurable en las propiedades del mismo. La salida se activa en cero cuando el valor del píxel es menor que el umbral definido, y en uno, en caso contrario.

La implementación de sistemas basados en microprocesadores y el uso de memorias compartidas, para cambiar los parámetros de los bloques de procesado, permiten desarrollar diseños configurables en tiempo de ejecución. El nivel de umbral óptimo puede ser uno de estos parámetros, ya que éste depende de la imagen a procesar.

El bloque de procesado, disponible en la biblioteca XSGImgLib, obtiene una salida por cada ciclo de reloj sin presentar latencia dentro del sistema.

3.3. Comparación de arquitecturas

El diseño de las arquitecturas de filtros de ordenamiento es muy utilizado en el preprocesado de la imagen. Por ello es necesario optimizar los diferentes bloques para maximizar la

cantidad de recursos en otras tareas del sistema. Las arquitecturas genéricas permiten el desarrollo de sistemas de procesado para diferentes ventanas, consumiendo un grupo considerable de recursos. Mientras tanto la arquitectura del filtro específico de mediana permite solucionar este inconveniente.

3.3.1. Filtros genéricos y específicos

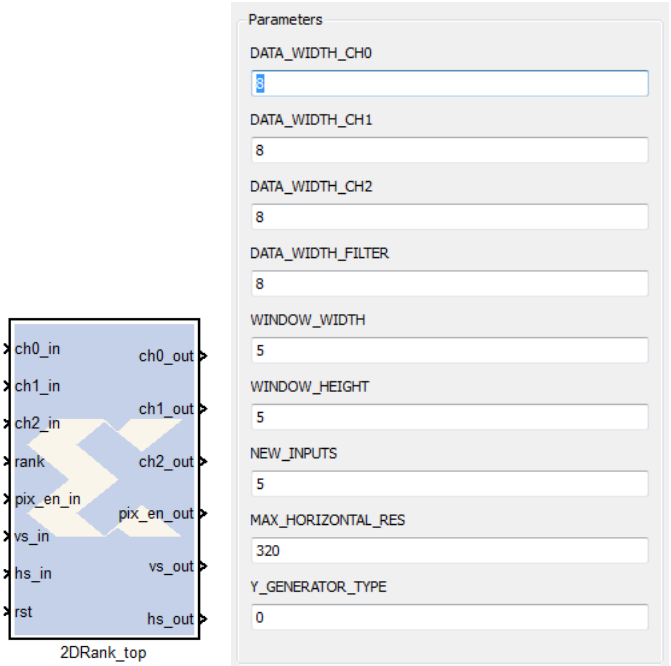
La biblioteca XSGImgLib provee un filtro de ordenamiento genérico, para ventanas de 3×3 y 5×5, y un filtro de mediana específico para una ventana de 3×3. Utilizando la arquitectura del filtro de mediana específico, como se describe en la Subsección 3.2.2, sólo pueden ser implementados sistemas para ventanas de 3×3 píxeles. El desarrollo de sistemas de procesado con diferentes tamaños de ventana, requiere la utilización de otras arquitecturas.

La comparación entre el filtro de ordenamiento genérico de 3×3 y filtro de mediana muestra el ahorro de recursos de esta arquitectura sobre la genérica. La Tabla 6.54 recoge los resultados de las comparaciones entre ambos bloques de la biblioteca para una placa de desarrollo *Spartan-3A DSP 1800* [34]. El filtro de mediana específico alcanza un ahorro de recursos del 35.18 % y una frecuencia cuatro veces mayor que en la versión genérica.

Su principal desventaja es la imposibilidad de exportar la arquitectura para ventanas de otros tamaños. El filtro de ordenamiento genérico de 5×5 logra un frecuencia de trabajo de 62.992 MHz y el consumo de recursos es cinco veces mayor que en la versión genérica de 3×3. Este aumento en la cantidad de recursos utilizados está dado por la necesidad de incrementar la cantidad (k) de registros S por procesador y sus tamaños $(k-1$ bits), así como los bits de los registros de posición (R) y de los bloques de cálculo.

3.3.2. Filtros de la biblioteca XSGImgLib y sistema de procesado utilizando VHDL

Como se menciona en capítulos anteriores, la biblioteca de procesamiento de imágenes XSGImgLib utiliza un flujo de diseño basado en modelos, presentando algunos inconvenientes comparados con los diseños en VHDL. Szedo publica en [26] una nota de aplicación para un sistema de procesamiento basado en la arquitectura de Chakrabarti de un filtro de ordenamiento genérico, mostrado en la Figura 3.12.



(a) Diagrama. **(b)** Propiedades.

Figura 3.12: Bloque de ordenamiento en VHDL.

Este sistema de procesamiento de imágenes en colores, altamente configurable, permite la selección de varios parámetros internos (Figura 3.12b) como: la cantidad de bits de los canales (de 1 a 16 bits en los campos *"DATA_WIDTH_CHX"*), el tamaño de la ventana (campos *"WINDOW_WIDTH"* y *"WINDOW_HEIGHT"*), la cantidad de píxeles de entrada en paralelo (campo *"NEW_INPUTS"*), la resolución máxima de la imagen en la horizontal (campo *"MAX_HORIZONTAL_RES"*), y la cantidad de bits del píxel de salida (campo *"DATA_WIDTH_FILTER"*) [26]. Además, el mismo realiza un cambio de modelo de representación de la imagen de RGB a YUV[3] o YCrCb. El sistema presenta siete entradas divididas en: tres canales de color ($ch0\_in - ch2\_in$), la posición del píxel deseado ($rank$) y las señales de sincronismo de vídeo (pix_en[4], hs[5] y vs[6]). Estas últimas señales son salidas del sistema conjunto con los píxeles procesados ($ch0\_out - ch2\_out$).

[3]Espacio de color típicamente usado como parte de un conducto de imagen en color definido en términos de una componente de luminancia y dos componentes de cromaticidad para la difusión de televisión.
[4]Habilitación del píxel.
[5]Sincronismo horizontal.
[6]Sincronismo vertical.

Aunque el modelo no está incluido en alguna biblioteca de procesamiento, el autor provee los códigos VHDL y un modelo de Simulink para realizar pruebas de simulación y co-simulación del bloque de procesado. Un sistema de procesamiento para imágenes en colores, comparable con el realizado en VHDL, es implementable utilizando tres bloques de procesamiento de la biblioteca XSGImgLib (uno para cada color) y un sistema de propagación para las señales de sincronismo.

El consumo de recursos para una placa *Spartan-3A DSP 1800* y una imagen de 320 píxeles de resolución horizontal, utilizando un sistema de procesado basado en el filtro de mediana para una vecindad de 3×3 y para el bloque en VHDL, se muestra en la Tabla 6.55. La Tabla 6.56 expone la comparación de los resultados de implementación utilizando un sistema que presenta como base un bloque de ordenamiento de 5×5 y un sistema en VHDL con la misma vecindad. Para la versión de 3×3 el sistema en VHDL logra una disminución de los recursos de un 64.13 % con una frecuencia de trabajo 1.37 veces mayor, mientras que, el ahorro de recursos para una vecindad de 5×5 es de un 56.85 % duplicando la frecuencia.

Los resultados alcanzados demuestran las posibilidades que presenta la programación en lenguajes de descripción de hardware sobre los diseños basados en modelos. Cabe resaltar que el sistema en VHDL utiliza dos bloques de memoria BRAM para la versión de 3×3 y cuatro para la de 5×5, mientras que los diseños de la biblioteca XSGImgLib evitan la utilización de estos recursos hardware.

Capítulo 4
Operadores morfológicos y bloques de control

LA morfología matemática (MM) es la teoría que estudia el análisis de las estructuras espaciales. Se le llama morfología porque se centra en el estudio de las formas de los objetos, y es matemática porque el análisis se basa en la geometría, el álgebra y varias teorías. La morfología matemática no es sólo una teoría, además es una potente técnica de procesamiento de imágenes [9, 17, 25].

Este capítulo desarrolla el estudio de los operadores morfológicos, como parte de la biblioteca de procesamiento de imágenes XSGImgLib, haciendo énfasis en su fundamento matemático y el funcionamiento de sus arquitecturas. Además se describen los bloques de la paleta de control de la biblioteca, necesarios para la construcción de los sistemas de procesado.

4.1. Operadores morfológicos

Las operaciones morfológicas sobre las imágenes son algoritmos que analizan la estructura geométrica de las mismas. Éstas pueden ser utilizadas tanto en imágenes binarias como en escalas de grises o en colores, y son muy útiles en diversas áreas de procesado como la detección de contornos, la restauración y el análisis de texturas. Un operador morfológico utiliza un elemento estructural (SE) para procesar la imagen. El cambio del SE permite la obtención de resultados diferentes utilizando la misma operación [7, 9, 15]. El Anexo 4 muestra los elementos estructurales básicos para ventanas de 3×3 y 5×5. Los resultados de la aplicación de estos operadores con diferentes SE se muestran en figuras posteriores.

Los operadores morfológicos pueden ser aplicados a imágenes binarias, en escalas de grises o en colores. La morfología matemática para imágenes en escalas de grises es más compleja y difícil de entender que la morfología para imágenes binarias. El concepto es el mismo pero en vez de tener un elemento estructural dentro de un objeto en dos dimensiones, éste debe estar contenido en un objeto de tres dimensiones [7, 17, 29].

Las imágenes cromáticas no presentan un orden natural para los colores en las coordenadas espaciales, pues son representados en vectores de tres componentes. La extensión de la morfología a imágenes en color requiere un estudio específico del modelo de representación de las mismas [17].

Las operaciones morfológicas de interés en este trabajo son las aplicadas a imágenes binarias. Éstas son convertidas desde su representación en escala de grises por procesos de detección de contornos y de umbralización.

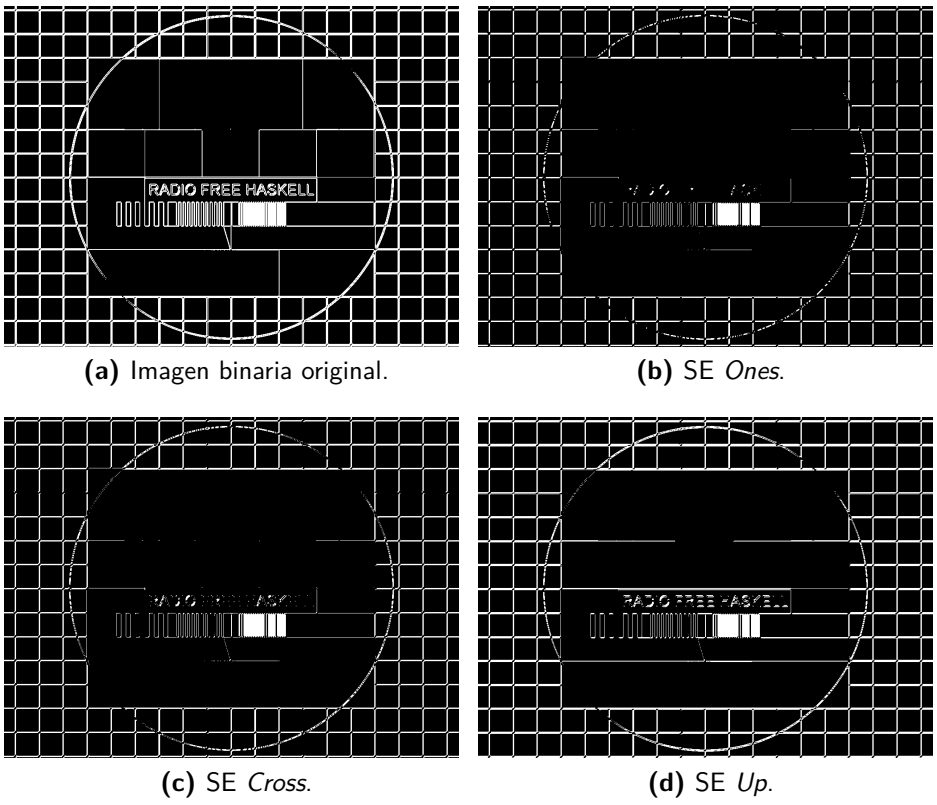
4.1.1. Fundamento matemático

Las operaciones básicas del procesado morfológico son la erosión y la dilatación [7, 29]. Usualmente la erosión es atribuida a encoger la imagen, mientras que la dilatación hace lo contrario. Los resultados de estas operaciones dependen del elemento estructural seleccionado para la misma.

La aplicación del operador erosión, como se muestra en la Figura 4.1, obtiene una imagen resultante, con un píxel en el primer plano por cada píxel de la imagen original que contenga al elemento estructural dentro del objeto delimitado, sin embargo, el operador dilatación obtiene un píxel del primer plano, correspondiente al origen del elemento estructural, por cada píxel de éste que contenga al objeto en la imagen, como se muestra en la Figura 4.2 [9, 15, 25].

La erosión de un objeto X en una imagen con un elemento estructural B, se denota por $\varepsilon_B(X)$ y se define como la localización de los puntos de la imagen (x) tales que el elemento estructural esté incluido en el objeto cuando su origen se encuentre en la posición x, como se muestra en la Ecuación 4.1. Esta ecuación puede ser reescrita en términos de intersección y de traslación determinada por el SE, como expone la Ecuación 4.2 [7, 25, 29].

$$\varepsilon_B(X) = \{x \mid B_x \subseteq X\}. \tag{4.1}$$



(a) Imagen binaria original. **(b)** SE *Ones*.

(c) SE *Cross*. **(d)** SE *Up*.

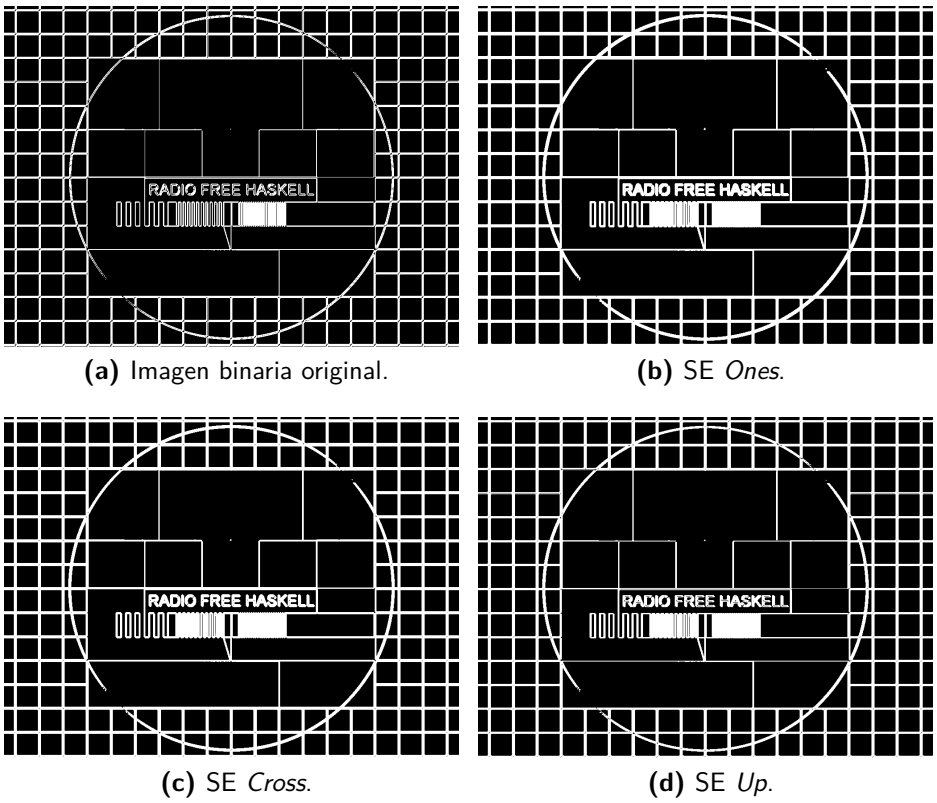
Figura 4.1: Resultados de la aplicación de operaciones de erosión con diferentes SE.

$$\varepsilon_B(X) = \bigcap_{b \epsilon B} X_{-b}. \tag{4.2}$$

Utilizando la notación actual, la erosión del objeto X por el elemento estructural B puede ser representada por la Ecuación 4.3, definida para todo valor de x donde $B + x$ esté contenido en el objeto [15].

$$X \Theta B = \{x : B + x < X\}. \tag{4.3}$$

Para imágenes binarias el resultado de la erosión es el crecimiento de las áreas negras en las mismas. Utilizando imágenes en escalas de grises, la erosión provoca el aumento de las áreas oscuras [15].



(a) Imagen binaria original. **(b)** SE *Ones*.

(c) SE *Cross*. **(d)** SE *Up*.

Figura 4.2: Resultados de la aplicación de operaciones de dilatación con diferentes SE.

La dilatación, como antes se ha mencionado, obtiene un resultado opuesto al anterior expuesto. Cuando un objeto X en una imagen es dilatado por un elemento estructural B, la operación $[\delta_B(X)]$ se define como la localización de los puntos de la imagen (x) cuando el elemento estructural toca al objeto, coincidiendo su origen con x, como describe la Ecuación 4.4. Ésta puede ser reescrita como la unión de traslaciones determinadas por el SE, utilizando la Ecuación 4.5 [7, 25, 29].

$$\delta_B(X) = \{x \mid B_x \cap X \neq 0\}. \tag{4.4}$$

$$\delta_B(X) = \bigcup_{b \epsilon B} X_{-b}. \tag{4.5}$$

Muchos autores denotan la dilatación utilizando términos actuales como se muestra en la Ecuación 4.6, definida en todo valor de b que sobresalga del objeto, siendo b parte del elemento estructural.

$$X \oplus B = \cup\{X + b : b \epsilon B\}. \tag{4.6}$$

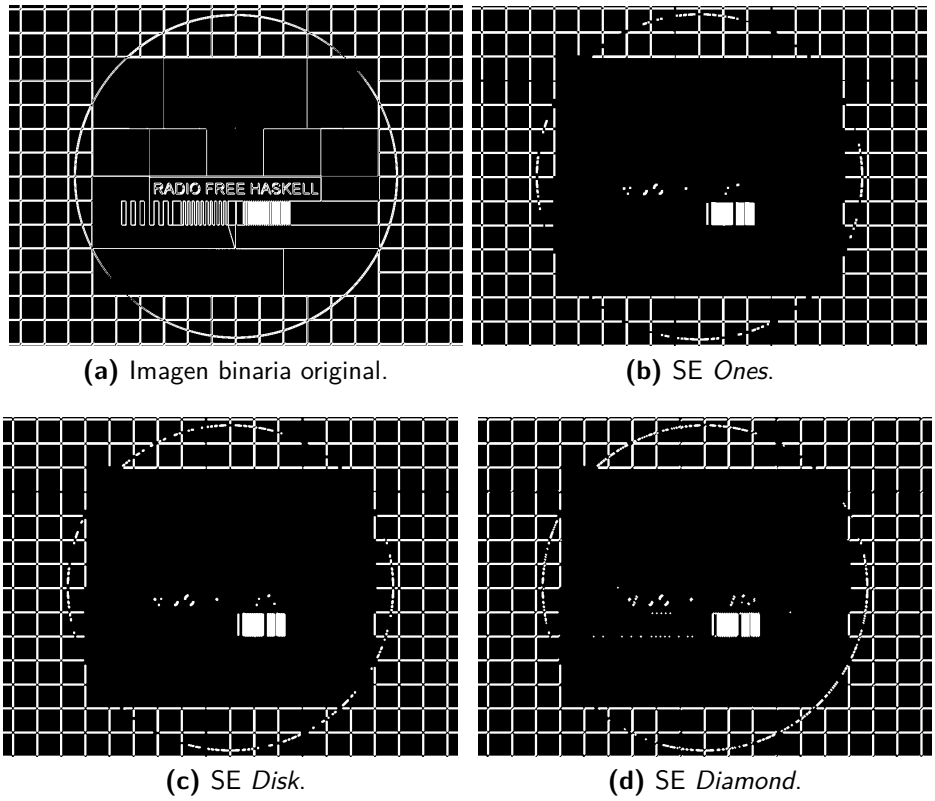
La dilatación provoca el aumento de las áreas blancas de las imágenes al utilizar morfología matemática binaria. Para imágenes en escalas de grises, se obtiene un crecimiento de las áreas claras en las mismas [15].

Una de las principales potencialidades de los operadores morfológicos es la posibilidad de obtener varios resultados con las operaciones básicas realizadas en diferente orden. Si la salida de una operación de erosión es dilatada el resultado es llamado apertura, mientras que la operación inversa, llamada cierre, es una de erosión luego de una dilatación. Estos dos operadores son muy utilizados en la restauración de imágenes. La Figura 4.3 y la Figura 4.4 muestran el resultado de las operaciones de apertura y cierre respectivamente con diferentes elementos estructurales para una ventana de 5×5.

La erosión no sólo elimina la información de la imagen que no contenga al elemento estructural, sino que también reduce todos los demás objetos. La aplicación de una dilatación, luego de la erosión permite eliminar la información innecesaria en la imagen para después ser modificada a su tamaño original. Un vez concluida la operación de erosión no es posible recobrar totalmente la imagen original mediante la transformación inversa. La idea del operador dilatación es poder recuperar la imagen tanto como sea posible, mediante la transformación del resultado del proceso de erosión [25].

La apertura (γ) de un objeto X por un elemento estructural B se denota por $\gamma_B(X)$ y se define como la erosión del objeto por el elemento estructural seguida de la dilatación por el SE reflectado ($\check{B}$), mostrada en la Ecuación 4.7. Aunque la apertura es denotada utilizando términos de erosión y dilatación, puede ser representada en notación de conjuntos como expone la Ecuación 4.8 [25].

$$\gamma_B(X) = \delta_{\check{B}}\left[\varepsilon_B(X)\right]. \tag{4.7}$$



(a) Imagen binaria original. **(b)** SE *Ones*.

(c) SE *Disk*. **(d)** SE *Diamond*.

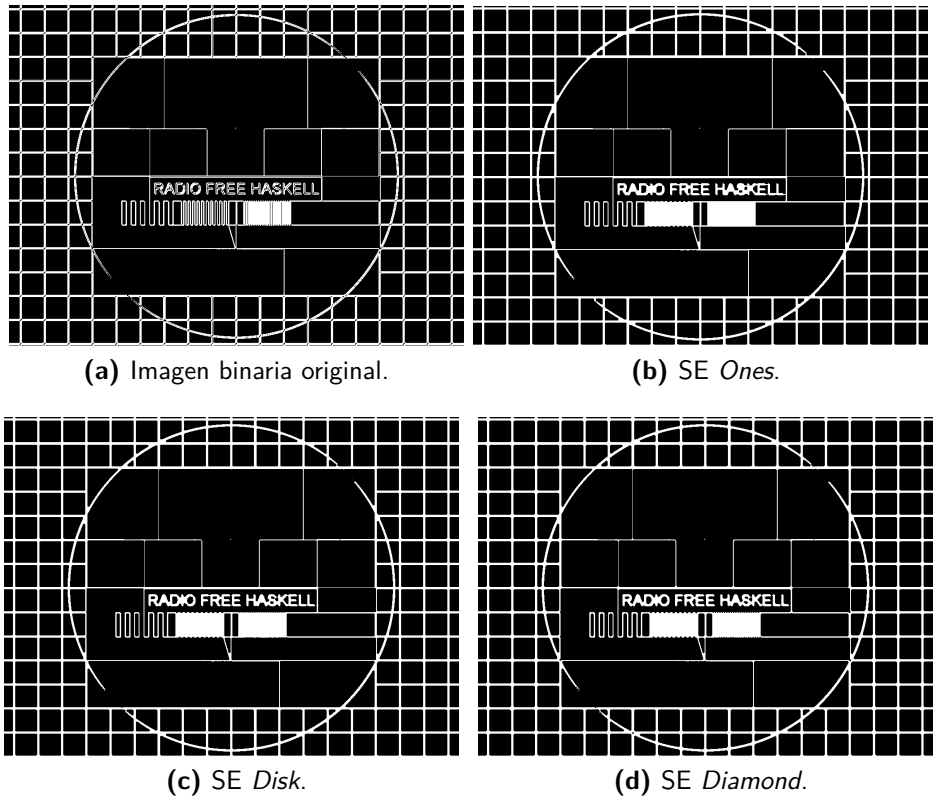
Figura 4.3: Resultados de la aplicación de operaciones de apertura con diferentes SE.

$$\gamma_B(X) = \bigcup_x \{B_x \mid B_x \subseteq X\}. \tag{4.8}$$

Utilizando la notación actual, la apertura puede ser representada como muestra la Ecuación 4.9.

$$X \circ B = (X \Theta B) \oplus B. \tag{4.9}$$

El cierre (ϕ) es la operación inversa a la apertura. Éste se aplica para recuperar la forma original de una imagen luego de ser dilatada. El cierre de un objeto X por medio de un elemento estructural B se denota por $\phi_B(X)$, y se define como la dilatación del objeto por el elemento estructural, seguida de la erosión por el SE reflectado ($\check{B}$), mostrada en la Ecuación 4.10. Ésta también puede ser denotada en teoría de conjuntos por la



(a) Imagen binaria original. **(b)** SE *Ones*.

(c) SE *Disk*. **(d)** SE *Diamond*.

Figura 4.4: Resultados de la aplicación de operaciones de cierre con diferentes SE.

Ecuación 4.11, y es usualmente escrita como expone la Ecuación 4.12 [25].

$$\phi_B(X) = \varepsilon_{\check{B}}\left[\delta_B(X)\right]. \tag{4.10}$$

$$\phi_B(X) = \bigcap_x \{B_x^c \mid X \subseteq B_x^c\}. \tag{4.11}$$

$$X \bullet B = (X \oplus B)\Theta B. \tag{4.12}$$

4.1.2. Arquitecturas hardware de procesamiento morfológico de imágenes de la biblioteca XSGImgLib

La definición matemática de la Subsección 4.1.1, para la erosión y la dilatación, es traducida a algoritmos de detección del valor mínimo o el valor máximo en la ventana, incidida por el elemento estructural. La Figura 4.5 muestra la operación morfológica realizada sobre una imagen. El algoritmo consiste en detectar el valor mínimo (erosión) o máximo (dilatación) de los píxeles de la ventana luego de eliminar todos los correspondientes a coeficientes nulos del elemento estructural. Para los operadores apertura y cierre se obtienen los valores mínimos y máximos según el orden definido en la Subsección 4.1.1 para los operadores erosión y dilatación.

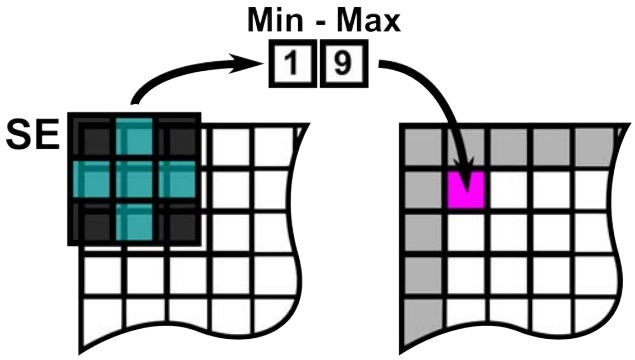


Figura 4.5: Algoritmo para los operadores morfológicos.

4.1.2.1. Operadores morfológicos erosión y dilatación

La operación erosión se compone por la obtención del valor mínimo entre los píxeles en la ventana, afectados por el elemento estructural. En cambio, la dilatación se basa en la obtención del valor máximo en las mismas condiciones. El SE es el encargado de seleccionar los valores de los píxeles a comparar. El diseño de módulos específicos para la obtención de mínimos y máximos permite la optimización de las arquitecturas de estas operaciones.

La arquitectura del bloque de erosión, mostrada en la Figura 4.6, utiliza $h-1$ bloques de registros (Z^{-1}) después de los almacenadores de línea para conservar los píxeles en la ventana. Luego de la paralelización de la información, se utilizan h módulos de obtención del valor mínimo con h entradas (Figura 4.7a), devolviendo el menor de los píxeles en la ventana afectados por el elemento estructural (*Min Block* SE_{COL1}- *Min Block* SE_{COLh}).

Por último, las h muestras mínimas de cada etapa de almacenamiento son pasadas a un módulo de obtención del valor mínimo (*Min Block NO SE*), sin ser afectadas por el SE (Figura 4.7b). El resultado de la salida de este bloque es el valor mínimo de los píxeles en la ventana, modificados por el elemento estructural.

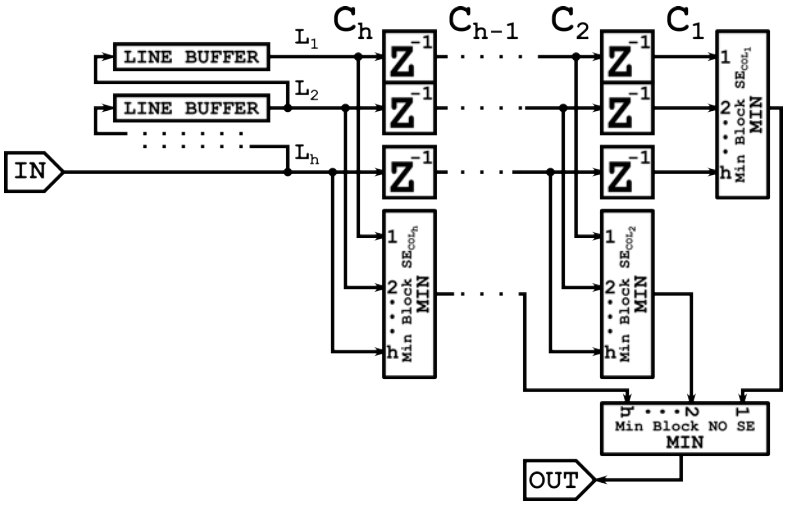
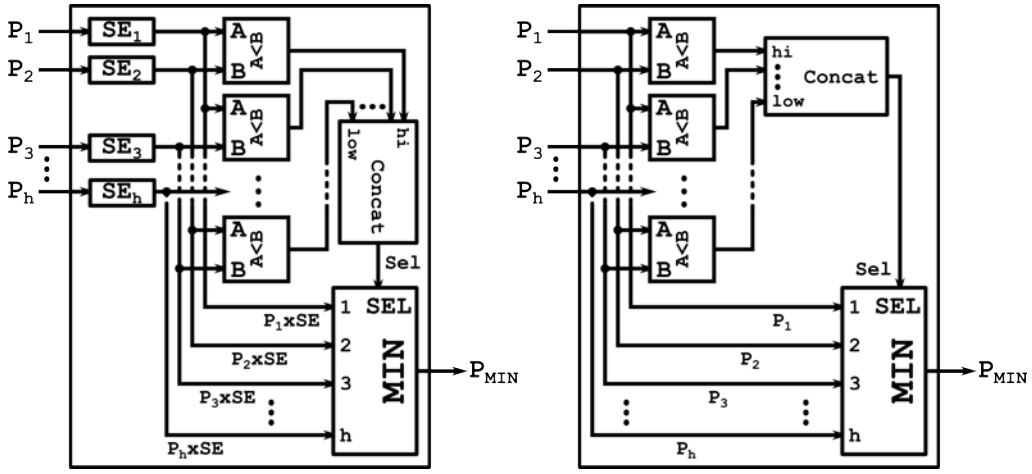


Figura 4.6: Arquitectura del bloque de erosión.

El bloque de obtención del valor mínimo, mostrado en la Figura 4.7a, se compone de h entradas ($P_1 - P_h$), una para cada píxel en la columna de procesado. Los valores nulos del SE en la columna de procesado saturan los píxeles correspondientes (maximizan su valor), utilizando bloques OR con el valor binario "1", mientras que los píxeles respectivos a los coeficientes unitarios mantienen su valor original. Esta operación asegura que estos píxeles no influyan en la obtención del mínimo valor en dicha columna. El resto de la arquitectura del componente coincide con el módulo de obtención del valor mínimo no modificado por el elemento estructural, donde los píxeles perturbados ($P_0 \times SE - P_h \times SE$) son introducidos al bloque de comparación.

El bloque de cálculo del valor mínimo (Figura 4.7b) se compone por $\frac{h(h-1)}{2}$ comparadores de dos entradas (*A*<*B*), que devuelven a la salida un valor binario (0 ó 1) según se cumpla la condición de comparación, por un bloque de concatenación (*Concat*), para formar un código binario (Sel) con los resultados de los bloques de comparación, que permita seleccionar el valor mínimo, y por un bloque de selección (*MIN*). La función de este último bloque es entregar el píxel mínimo de las entradas ($P_1 - P_h$) de acuerdo con el código obtenido del bloque de concatenación. Su composición cuenta con un multiplexor de $2^{\frac{h(h-1)}{2}}$ entradas

codificadas según las diferentes combinaciones de salidas disponibles. La salida de este multiplexor coincide con el valor mínimo de los píxeles a comparar. En el caso específico de la ventana de 5×5, para reducir el número de bloques de comparación y la cantidad de entradas del multiplexor, se realiza la operación con tres valores, obteniendo el píxel mínimo de ellos. Paralelamente se ejecuta la comparación de los dos píxeles restantes en la columna para luego realizar la misma operación con ambas salidas parciales. El bloque de obtención del valor mínimo, utiliza multiplexores en cascada, para obtener el resultado entre todas las entradas.



(a) Mínimo con SE. **(b)** Mínimo sin SE.

Figura 4.7: Estructuras básicas del operador erosión.

La arquitectura de operador dilatación, mostrada en la Figura 4.8, como bien se menciona anteriormente, funciona de la misma manera que el operador erosión. Su diferencia radica en los módulos básicos, que en su lugar devuelven el valor máximo (*Max Block* SE_{COL1}-*Max Block* SE_{COLh}). El primer módulo (Figura 4.9a) se ve afectado por la incidencia de una parte de los coeficientes del elemento estructural. Los píxeles correspondientes a los valores nulos del SE son modificados por un operador AND con el valor "0", manteniendo los píxeles relacionados con los coeficientes unitarios. El resto de la operación concuerda con el módulo de obtención del valor máximo no modificado (Figura 4.7b). La estructura y funcionamiento de este bloque es similar a la descrita en el componente de obtención del valor mínimo cambiando la función de los comparadores.

Los bloques de erosión y dilatación se incluyen en la biblioteca de procesado de imágenes XSGImgLib para ventanas de 3×3 y 5×5 con posibilidades de seleccionar el elemento

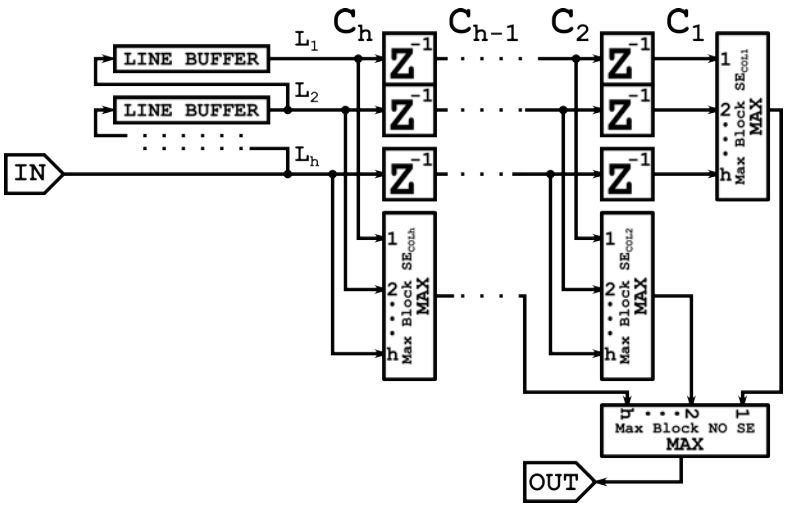


Figura 4.8: Arquitectura del bloque de dilatación.

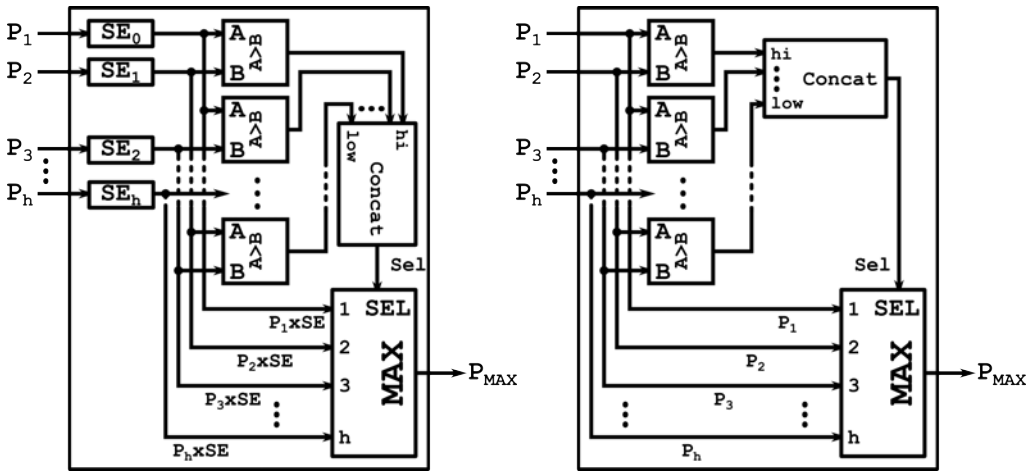
estructural. La Figura 4.10 muestra las propiedades básicas de estos bloques.

Al tener arquitecturas similares, los bloques de erosión y dilatación presentan consumo de recursos idénticos para un mismo tamaño de la ventana. La Tabla 6.28 expone el consumo de recursos de los bloques de erosión o dilatación para ventanas de 3×3 y 5×5. Basándose en la Tabla 6.24 y la Tabla 6.25, la utilización de bloques de ordenamiento genéricos para implementar los operadores erosión o dilatación en una placa *Spartan-3A DSP 1800*, requiere 30 veces más recursos con 7 veces menos frecuencia de trabajo para una ventana de 3×3, mientras que para una vecindad de 5×5, el consumo de recursos se eleva en casi 50 veces reduciendo la frecuencia de trabajo en casi 6 veces.

La latencia del sistema está dada por la demora en los almacenadores de línea (τ_{lb}), definida en la Ecuación 4.13, y las etapas de almacenamiento de la operación (τ_{op}) definida en la Tabla 6.10. Este valor es calculable con la Ecuación 2.5.

4.1.2.2. Operadores morfológicos apertura y cierre

Las operaciones de apertura y de cierre se basan en los bloques erosión y dilatación descritos anteriormente. La arquitectura de estos bloques está dada por la realización de operaciones de erosión y de dilatación una a continuación de la otra. La operación de apertura presenta en su diseño (Figura 4.11) un bloque de erosión seguido de un bloque de almacenadores de línea para la paralelización de la salida serie del bloque anterior. A continuación se aplica



(a) Máximo con SE. **(b)** Máximo sin SE.

Figura 4.9: Estructuras básicas del operador erosión.

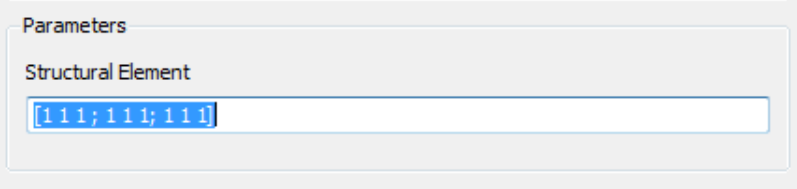


Figura 4.10: Propiedades de los bloques erosión y dilatación.

un operador dilatación obteniendo el resultado. En cambio, el bloque de cierre intercambia el orden de los operadores dilatación y erosión, como muestra la Figura 4.12.

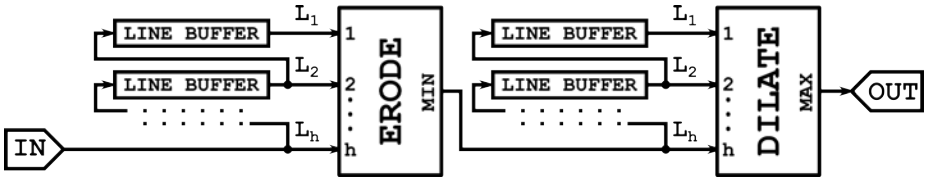


Figura 4.11: Arquitectura del bloque apertura.

El consumo de recursos es idéntico en ambas arquitecturas y está dado por la cantidad de columnas de la imagen a procesar y por el tamaño de la ventana. La Tabla 6.29 presenta los resultados de implementación de estos bloques en una *Spartan-3A DSP 1800* para una imagen de 64 columnas. Los parámetros de los mismos incluyen, además del elemento estructural, la configuración del bloque de almacenadores de línea entre los operadores de procesado. La Figura 4.13 muestra la ventana de propiedades de los mismos.

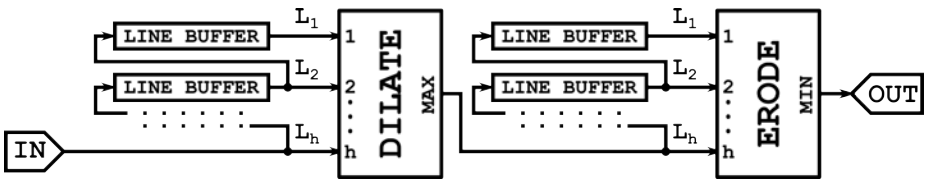


Figura 4.12: Arquitectura del bloque cierre.

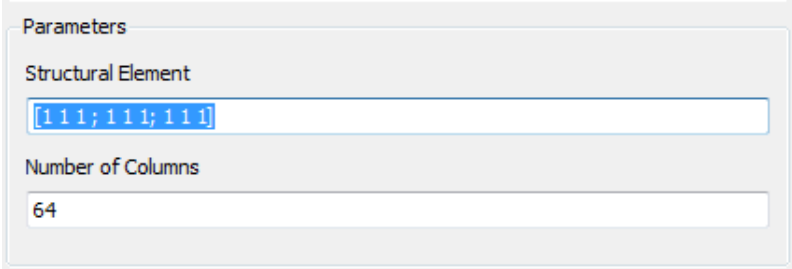


Figura 4.13: Propiedades de los bloques apertura y cierre.

La latencia del sistema se define en la Ecuación 2.5, dada por la demora en los almacenadores de línea (τ_{LB}) de la Ecuación 4.13 y la latencia interna de la operación (τ_{op}) de la Tabla 6.10. Esta demora está condicionada por la cantidad de columnas de la imagen y el almacenamiento interno de las operaciones.

4.2. Paleta de control de la biblioteca XSGImgLib

La paleta de control de la biblioteca XSGImgLib contiene bloques que permiten la implementación de los sistemas de procesado. Actualmente la paleta cuenta con dos bloques encargados de estas funciones, tanto para ventanas de 3×3 como de 5×5. En futuras actualizaciones de la biblioteca, la paleta contará con bloques para el manejo de las señales de vídeo, así como bloques para la conversión de espacios de colores, agregándole mayor funcionalidad a la misma.

Los bloques almacenadores de línea y los registros, como se menciona en la Subsección 1.3.3, son los encargados de la conversión de la información serie a paralelo, para explotar las capacidades del hardware. El procesado basado en una vecindad es el causante de esta necesidad. Los píxeles llegados desde la fuente son enviados al sistema de procesado uno a uno, por lo que es necesario mantener una cantidad de píxeles, determinados por el tamaño de la ventana, para ejecutar los diferentes algoritmos paralelos de procesado. Los bloques

almacenadores de línea se encargan de conservar los píxeles de $h-1$ filas ($L_1 - L_{h-1}$) de la imagen para ser entregados a los registros. Éstos a su vez mantienen los valores de $h-1$ columnas ($C_1 - C_{h-1}$) obteniendo así una ventana de procesado de $h \times h$ píxeles. La fila más reciente viene directamente desde la entrada (L_h) y la columna restante se toma desde los almacenadores de línea (C_h).

4.2.1. Almacenadores de línea

La arquitectura de los almacenadores de línea, expuesta en la Figura 4.14, utiliza $h-1$ registros de desplazamiento direccionables para su función. La capacidad de almacenamiento de estos registros depende de la resolución de la imagen en el eje horizontal, parametrizable en la ventana de propiedades del bloque como muestra la Figura 4.15. La Tabla 6.30 describe el consumo de recursos de la implementación de un bloque de almacenadores de línea para diferentes vecindades.

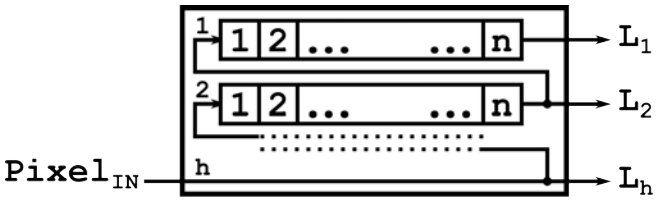


Figura 4.14: Arquitectura del bloque de almacenadores de línea.

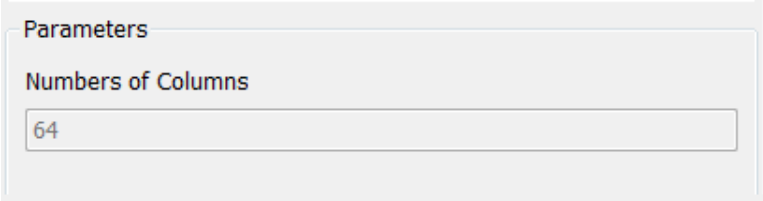


Figura 4.15: Propiedades del bloque de almacenadores de línea.

En cada ciclo de reloj un nuevo píxel es entregado por el subsistema de adquisición al bloque de almacenadores de línea devolviendo a su salida h píxeles ($L_1 - L_h$), que se corresponden con una nueva columna a procesar. Este procedimiento implica una latencia inicial en el sistema de procesado debido a la demora en completar este bloque. La Ecuación 4.13 expone el valor de esta latencia, donde rw es el radio de la ventana, definida en la Sección 2.1, y

n es la cantidad de píxeles de la imagen en el eje horizontal.

$$\tau_{lb} = 2rw \times n. \tag{4.13}$$

System Generator brinda dos bloques con funciones idénticas al descrito en esta subsección. El bloque *Virtex Line Buffer* permite construir subsistemas almacenadores de línea para la paralelización de la información. Su arquitectura utiliza memorias dedicadas (BRAM) para cada línea de procesado. El bloque, como muestra la Tabla 6.57 para imágenes de 64 columnas, presenta un ahorro de recursos lógicos del 56 %, en ambos tamaños de ventana, y la frecuencia de ejecución no se ve afectada para una vecindad de 3×3, mientras que disminuye en 60 MHz para una ventana de 5×5.

El bloque *Virtex2 Line Buffer* está optimizado para su uso en un FPGA de altas prestaciones de Xilinx, especialmente un *Virtex 2* [37]. La arquitectura de este bloque es similar a la anterior. Cuenta con bloques BRAM para almacenar las filas de la imagen y utiliza un consumo de recursos un 75 % menor que el bloque de la biblioteca XSGImgLib. La frecuencia de trabajo aumenta 35 MHz para una ventana de 3×3 y 10 MHz para una de 5×5. Los resultados de las comparaciones se muestran en la Tabla 6.58 para una placa de desarrollo de prestaciones medias. Estos resultados pueden variar considerablemente para otros FPGA.

Aunque los bloques de la biblioteca XSGImgLib presentan un mayor consumo de recursos con respecto a los diseñados por System Generator, éstos son una opción cuando el diseño del sistema de procesado necesita de bloques de memoria dedicados para su funcionamiento.

4.2.2. Registros

Los bloques de registros de la paleta de control de la biblioteca XSGImgLib forman parte del sistema de paralelización de la información. Su arquitectura, mostrada en la Figura 4.16, utiliza $h(h-1)$ bloques de registros genéricos (Z^{-1}) brindados por XSG. Su función principal es mantener las diferentes columnas de píxeles para las entradas de los bloques de procesado de la imagen (L_aC_b, donde $1 \leq a \leq h$ y $1 \leq b \leq h$).

Muchas arquitecturas de procesado, expuestas con anterioridad, incluyen este bloque dentro de las mismas. Sólo los bloques de convolución genéricos no agregan este componente a

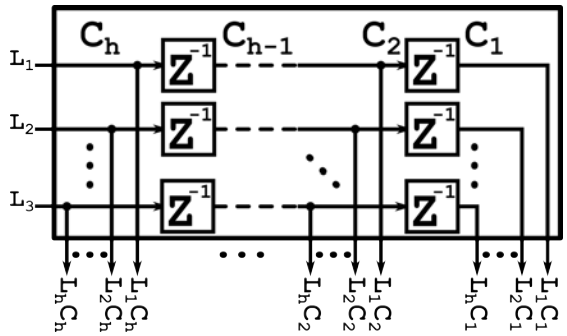


Figura 4.16: Arquitectura del bloque de registro.

sus estructuras, en aras de realizar diseños que permitan probar nuevos bloques de control, con el mismo subsistema de procesado. Las arquitecturas que requieren estos bloques presentan una latencia inicial determinada por la demora en ocupar totalmente el mismo. La Ecuación 4.14 expone este valor.

$$\tau_{rg} = 2rw. \tag{4.14}$$

Capítulo 5
Procesado de imágenes utilizando XSGImgLib

LA segmentación de imágenes puede ser definida como la división de las mismas en regiones que cumplan ciertas propiedades. En una imagen segmentada el elemento principal no es el píxel, sino la unión de varios de éstos. Esta técnica permite realizar mediciones de las diferentes regiones y obtener relaciones entre las adyacentes, siendo ésta muy utilizada en la interpretación cuantitativa de los datos en una imagen [13, 14, 25, 31].

Este capítulo describe la integración de los elementos de la biblioteca XSGImgLib en el desarrollo de un sistema de procesado de visión para la segmentación de imágenes.

5.1. Segmentación de imágenes

En términos matemáticos, la segmentación de una imagen A es la división de su dominio de definición (D_A) en n segmentos no nulos $(X_1, X_2, \ldots, X_n)$, cuya unión conforma D_A. Una imagen segmentada muestra, usualmente, los bordes de la misma, posibilitando su representación en valores binarios [25].

La segmentación de imágenes se basa en dos propiedades básicas de los valores de intensidad: la discontinuidad y la similitud. En la primera categoría el procesado se realiza dividiendo la imagen analizando los cambios abruptos en la intensidad de los píxeles. El principal algoritmo de segmentado es el clasificado en la segunda categoría, basándose en la división de la imagen según su similitud con un grupo de criterios definidos. La umbralización, el crecimiento, separación y unión de regiones son métodos de esta categoría [9].

La umbralización, descrita en la Sección 3.1, por su simplicidad de implementación ocupa una posición importante en la segmentación de imágenes. Una forma de extraer objetos de una imagen puede ser realizada utilizando un nivel adecuado de umbral (T). Cualquier punto (x, y) cuyo valor sea mayor que el nivel de umbral ($f(x, y) \geq T$) se define como un punto del objeto, mientra que de lo contrario se define como un punto del fondo [9].

La segmentación de imágenes basada en regiones utiliza los métodos de dilatación, división e unión de regiones. La dilatación es un procedimiento que agrupa píxeles o subregiones conformando regiones más amplias. La aproximación básica de este método consiste en conformar un grupo de píxeles, llamado semilla, y dilatar la región añadiendo esta semilla a los puntos colindantes que presenten propiedades similares a ésta [9].

Otra técnica alternativa, es subdividir, inicialmente, la imagen de forma arbitraria en regiones discontinuas para luego unirlas o dividirlas de acuerdo a la satisfacción de las condiciones. Empezando por la imagen completa, ésta se va dividiendo en regiones y subregiones mientras no se encuentren objetos con las propiedades buscadas. Sólo aplicando la división, la disección final contendrá regiones adyacentes con propiedades similares. Este problema se soluciona aplicando la unión de estas regiones adyacentes. El proceso debe terminar cuando no sea posible aplicar otra operación de división o de unión [9].

Para imágenes binarias la segmentación se refiere a la extracción de los objetos de interés conectados, así como la separación de objetos solapados. Utilizando las operaciones morfológicas de dilatación y erosión en imágenes binarias puede realizarse la extracción de los bordes claros de los objetos en el fondo oscuro de las mismas [31]. Este método de segmentado es aplicado en el ejemplo descrito en la Sección 5.2.

5.2. Segmentación de imágenes utilizando XSGImgLib

El ejemplo, presentado en esta sección, desarrolla la segmentación de imágenes aplicando operadores morfológicos. El sistema, presentado en la Figura 5.1, se divide en varias etapas de procesado, cada una con funciones definidas. Estas etapas son descritas posteriormente.

La imagen original, en modelo de representación RGB (Figura 5.2), se convierte a escala de grises utilizando las instrucciones de trabajo con imágenes de MATLAB. El resultado de la conversión se muestra en la Figura 5.3, donde se transforman los canales de colores de la imagen original a escala de grises, para enviarla a las unidades de procesado. Cada etapa

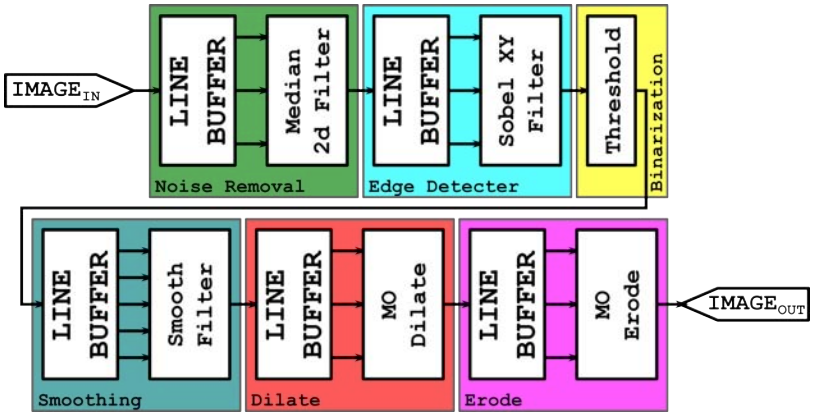


Figura 5.1: Sistema de procesado.

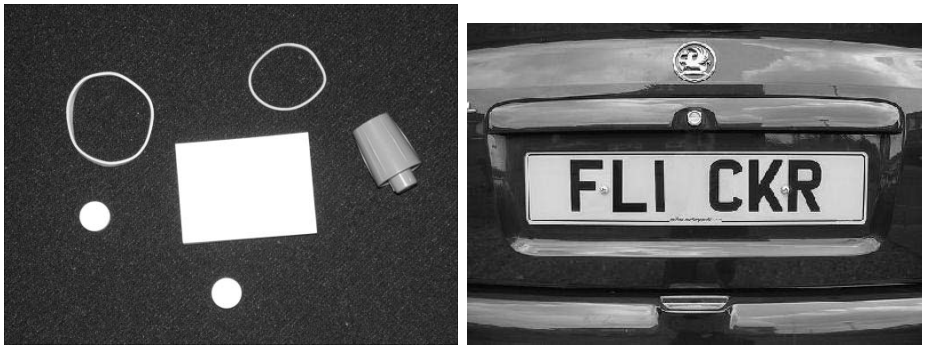
(a) Imagen 1. **(b)** Imagen 2.

Figura 5.2: Imagen original.

de procesado se concatena con la siguiente, devolviendo un píxel de salida en cada ciclo de reloj. El uso de bloques almacenadores de línea en las etapas de procesado es necesario para lograr el paralelismo de la información.

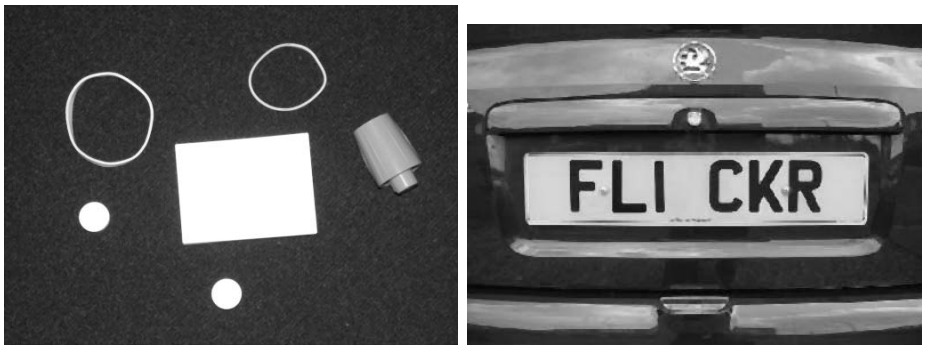
El sistema de procesado en su conjunto ocupa un 7.12 % de recursos del FPGA, un 17.26 % de los bloques de multiplicación dedicados (DSP48A) y presenta una frecuencia de trabajo de 33.612 MHz, para una placa *Spartan-3A DSP 1800*, como muestra la Tabla 6.31. En las subsecciones posteriores se describe el consumo de recursos de cada etapa, basándose en el total utilizado por el sistema. El mayor empleo de recursos en el mismo es ocupado en los bloques almacenadores de línea, pudiendo ser sustituidos por los *Virtex2 Line Buffer* de System Generator. Esta operación implicaría el uso de bloques BRAM en el sistema.

El diseño, para una imagen con una resolución de 512×384 píxeles, permite el procesado de cada píxel en 29.743 ns, y de un fotograma en 5.85 ms (171.005 Hz). La latencia del



(a) Imagen 1. **(b)** Imagen 2.

Figura 5.3: Imagen de entrada.



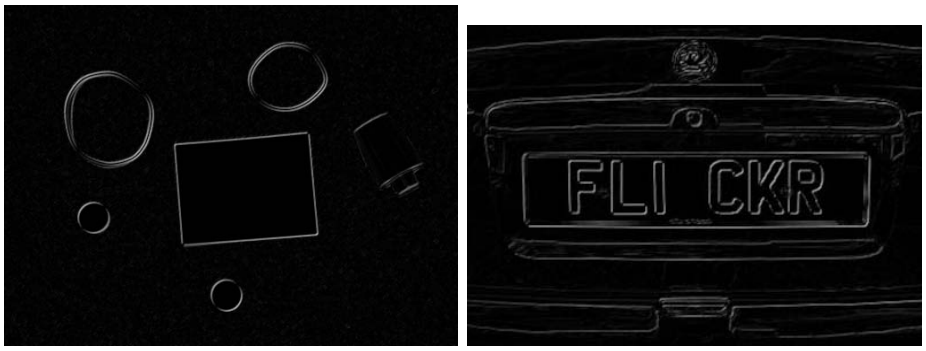
(a) Imagen 1. **(b)** Imagen 2.

Figura 5.4: Imagen sin ruido.

sistema se calcula por la suma de las demoras en cada bloque de procesado, teniendo como resultado 6159 ciclos de reloj para obtener la primera salida válida, para un total de 0.183 ms de espera.

5.2.1. Eliminación de ruido

La etapa de reducción de ruido es el primer paso del sistema de procesado. Esta etapa se implementa con un filtro de mediana, luego de la paralelización de la información mediante los almacenadores de línea. La Tabla 6.32 muestra el consumo de recursos en esta etapa (41.99 %), de acuerdo con el total utilizado en el sistema. La Figura 5.4 expone el resultado de la aplicación de este procesamiento en el sistema, eliminando el ruido impulsivo en la imagen.



(a) Imagen 1. **(b)** Imagen 2.

Figura 5.5: Contornos de la imagen.

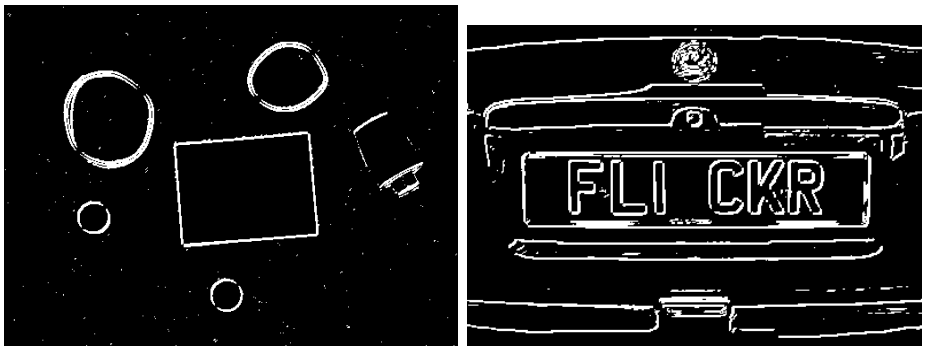
5.2.2. Detección de bordes y umbralización

La etapa siguiente se encarga de la detección de bordes de la imagen sin ruido, seguida por la etapa de representación binaria. La detección de contornos (Figura 5.5) se realiza utilizando un filtro específico *Sobel X-Y*, normalizando la salida con la suma de los elementos positivos del kernel. Para la normalización se utilizan bloques lógicos de System Generator fuera de la unidad de procesado. La Tabla 6.33 muestra el consumo de recursos de la etapa de detección de bordes (27 %).

Luego de esta operación la imagen es enviada al bloque de umbralización para convertirla en binaria (Figura 5.6). El valor de umbral es calculado previamente en MATLAB y ajustado en tiempo de diseño. Con la incorporación de un procesador embebido, es posible configurar este valor en tiempo de ejecución, mejorando la adaptabilidad del sistema de procesado. Esta operación, al ser puntual, no necesita de la paralelización de la información, y de acuerdo a su arquitectura no consume una gran cantidad de recursos, como muestra la Tabla 6.34 (0.17 %).

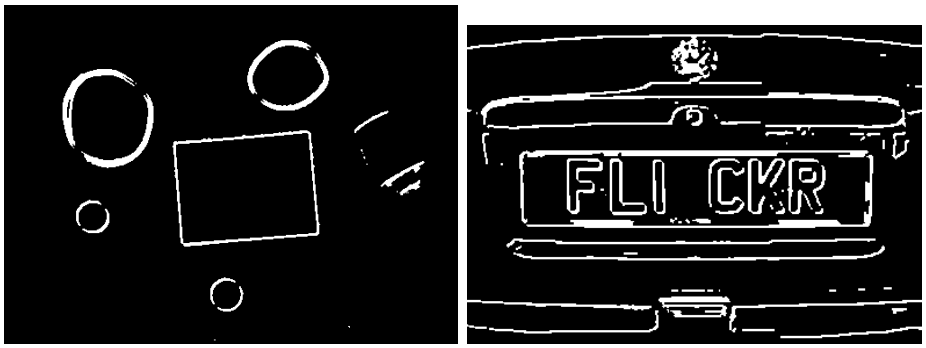
5.2.3. Suavizado de la imagen

La imagen binaria es filtrada por un proceso de suavizado tipo *Smooth*, implementado sobre un bloque genérico de 5×5. Las entradas de este bloques se encuentran optimizadas para el trabajo con imágenes binarias reduciendo así el consumo de recursos, como muestra la Tabla 6.35. El 100 % de los bloques de multiplicación del sistema de procesado se utilizan en esta etapa, así como el 42.41 % de los bloques lógicos. El resultado de este proceso,



(a) Imagen 1. **(b)** Imagen 2.

Figura 5.6: Contornos binarizados.



(a) Imagen 1. **(b)** Imagen 2.

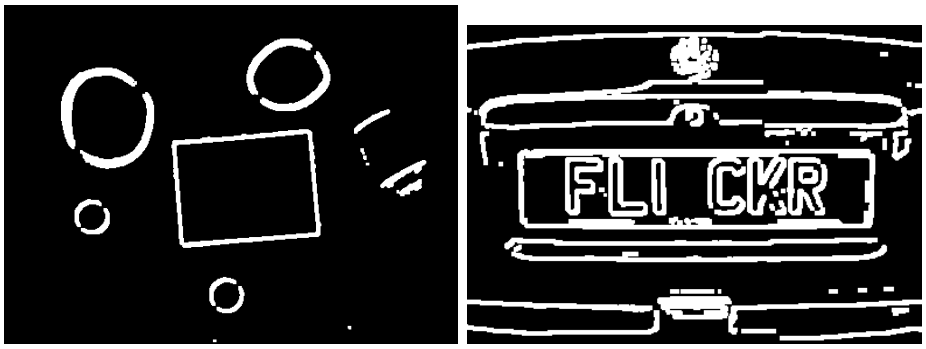
Figura 5.7: Imagen binaria suavizada.

expuesto en la Figura 5.7, desenfoca la imagen, reduciendo el tamaño de las zonas aisladas en la misma.

5.2.4. Operaciones morfológicas

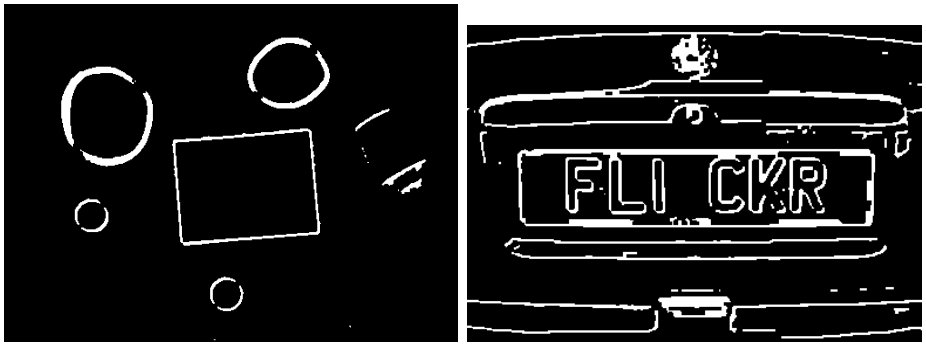
El último paso de este sistema lo componen las operaciones morfológicas sobre la imagen suavizada. Primeramente la etapa de dilatación, utilizando un elemento estructural con una ventana de 3×3 con todos sus coeficientes en "1" (*Ones*, ver Anexo 4), aumenta el tamaño de las zonas blancas de la imagen. Esta operación permite unir los puntos que formaban líneas en la imagen original y fueron eliminados por las anteriores etapas, como muestra la Figura 5.8.

Una vez realizado este paso, el resultado de la dilatación es erosionado para recuperar, tanto como sea posible, el tamaño original de los objetos en la imagen. Este proceso se realiza



(a) Imagen 1. **(b)** Imagen 2.

Figura 5.8: Imagen dilatada.



(a) Imagen 1. **(b)** Imagen 2.

Figura 5.9: Imagen segmentada.

utilizando un bloque de erosión de la biblioteca con el mismo SE de la etapa anterior. La Figura 5.9 muestra el resultado del proceso de segmentación de la imagen. La Tabla 6.36 y la Tabla 6.37 describen el consumo de recursos de estas operaciones según la utilización de recursos del sistema de procesado (3.42 % y 3.34 % respectivamente).

En el proceso de segmentación puede ser incluida una etapa intermedia entre los operadores morfológicos. Esta etapa se encarga de rellenar los píxeles que se encuentran dentro de los bordes de los objetos en la imagen dilatada. La misma no fue implementada en el ejemplo puesto que este bloque no se encuentra disponible en la biblioteca XSGImgLib.

Conclusiones

EL paralelismo de los algoritmos de procesado de imágenes permite el desarrollo de sistemas empotrados más eficientes que los algoritmos secuenciales sobre procesadores de propósito general. El diseño de aplicaciones para FPGA no sólo explota este paralelismo sino también la portabilidad del diseño y el bajo consumo de potencia del mismo, posibilitando el desarrollo de sistemas modulares y autónomos.

Las herramientas electrónicas de diseño automatizado basadas en modelos permiten la integración de diferentes plataformas de desarrollo, para los actuales flujos de diseño. Las técnicas de desarrollo basadas en lenguaje de descripción de hardware permiten un mejor aprovechamiento de los recursos de los dispositivos, en comparación con los diseños basados en modelos.

La implementación de módulos de procesado de imágenes parametrizables para System Generator, permite el desarrollo de sistemas basados en modelos con un elevado nivel de eficiencia. La biblioteca de procesado de imágenes desarrollada, posibilita la configuración de los diseños genéricos con diferentes parámetros de optimización y la selección de los bloques específicos, logrando unidades de procesado versátiles al alcance del usuario.

La definición matemática de la operación de convolución bidimensional, base del procesado lineal de imágenes, presenta total independencia de los píxeles en la ventana, permitiendo el paralelismo del algoritmo. Las versiones simétricas de los bloques genéricos de convolución, posibilitan un ahorro considerable de los recursos dedicados del FPGA sobre la versión no simétrica. De acuerdo con las características del procesado, el usuario puede seleccionar entre bloques con las operaciones de cálculo normalizadas y bloques a total precisión. Estos últimos, utilizan una mayor cantidad de recursos lógicos, mientras que los primeros reducen el consumo de recursos a costa de un error, resultado de las operaciones de cálculo del algoritmo, que está ligado a la parametrización de la precisión. Optimizando el sistema en área o velocidad permite establecer un adecuado compromiso entre la velocidad de

procesado y el consumo de recursos. Los diseños de filtros específicos de la biblioteca brindan una adecuada optimización de los recursos y de la velocidad de ejecución, siendo la mejor opción cuando se utilizan kernels predefinidos.

El procesamiento no lineal con XSGImgLib utiliza filtros de ordenamiento. La arquitectura específica mejora el consumo de recursos de los sistemas, pero sólo puede ser utilizada en ventanas de 3×3. El bloque de ordenamiento genérico permite modificar la posición del píxel de salida para el diseño de operadores morfológicos. La arquitectura utilizada para este bloque presenta un menor consumo de recursos al ser desarrollada con lenguajes de descripción de hardware. La implementación de operadores morfológicos, utilizando bloques específicos de posición mínima y máxima, permite la reducción considerable de los recursos del sistema.

La biblioteca XSGImgLib posibilita el diseño de sistemas de procesado de imágenes utilizando la concatenación de sus bloques. La utilización de bloques de biblioteca de System Generator para la implementación de los almacenadores de línea es una opción viable para la reducción de los bloques lógicos del sistema, aunque aumenta el consumo de bloques BRAM en el mismo.

Recomendaciones

EL diseño de la biblioteca de procesado, así como su aplicación en simulación y co-simulación originan las siguientes recomendaciones:

- Desarrollar nuevos bloques de procesado para la biblioteca XSGImgLib que permitan la utilización de la misma en otros procesos de filtrado de imágenes, no disponibles en la actualidad.
- Incorporar a la paleta de control bloques de manejo de señales de sincronismo, extendiendo al campo del procesado de vídeo las unidades desarrolladas.
- Utilizar nuevas arquitecturas de ordenamiento genérico, así como mejorar las ya existentes, para un reducir el consumo de recursos de los sistemas de procesado.
- Utilizar bloques básicos programados en lenguajes de descripción de hardware para potenciar las unidades de procesado.
- Desarrollar sistemas de procesado utilizando procesadores embebidos para la implementación de diseños adaptativos en tiempo de ejecución.

Bibliografía

A

[1] A M S Adario, E L Roehe, and S Bampi. Dynamically reconfigurable architecture for image processor applications. *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pages 623–628, 1999.

[2] Altera. Video and Image Processing Design Using FPGAs. *Altera White Papers*, (March):1–6, 2007.

B

[3] K Benkrid and S Belkacemi. Design and implementation of a 2D convolution core for video applications on FPGAs. *Third International Workshop on Digital and Computational Video, 2002. DCV 2002. Proceedings.*, (November):85–92, 2002.

[4] K Benkrid, D Crookes, J Smith, and A Benkrid. High Level Programming for FPGA Based Image and Video Processing using Hardware Skeletons. In *9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 1–8, 2001.

[5] Sheetal U Bhandari, Shashank S. Pujari, Shaila Subbaraman, and Rashmi Mahajan. Real Time Video Processing on FPGA Using on the Fly Partial Reconfiguration. In *2009 International Conference on Signal Processing Systems*, pages 245–247, 2009.

[6] M Bramberger, M Quaritsch, T Winkler, B Rinner, and H Schwabach. Integrating Multi-Camera Tracking into a Dynamic Task Allocation System for Smart Cameras. *IEEE Conference on Advanced Video and Signal Based Surveillance, 2005*, pages 474–479, 2005.

C

[7] Jesús Cáceres Tello. La visión artificial y las operaciones morfológicas en imágenes binarias. pages 1–7, 2002.

[8] C Chakrabarti. High sample rate array architectures for median filters. *IEEE Transactions on Signal Processing*, 42(3):707–712, 1994.

G

[9] Rafael C González and Richard E Woods. *Digital image processing*, chapter 2, 3, 4, 6, pages 66–70, 116–134, 205–208, 283–302, 308–313, 519–560. Prentice Hall, 2nd edition, 2002.

H

[10] S H Hajimowlana, G.a. Jullien, R Muscedere, and J W Roberts. Efficient pre-processing algorithms for an FPGA based in-camera video-stream processing system for industry inspection. *Canadian Conference on Electrical and Computer Engineering. Engineering Innovation: Voyage of Discovery. Conference Proceedings*, pages 835–838, 1997.

I

[11] Texas Instruments. Implementation of an Image Processing Library for the TMS320C8x (MVP). Data Sheet, 1997.

J

[12] C T Johnston, D G Bailey, and P Lyons. A Visual Environment for Real-Time Image Processing in Hardware. *EURASIP Journal on Embedded Systems*, 2006:1–8, 2006.

K

[13] Rafal Kluszczynski, Marie-Colette van Lieshouy, and Tomasz Schreiber. An Algorithm for Binary Image Segmentation Using Polygonal Markov Fields. *ICIAP*, pages 383–390, 2005.

M

[14] Javier Montenegro Joo. Image Segmentation through Encapsulation of its Constituents. *Revista de la Facultad de Ingeniería Industrial*, 13(1):77–79, 2010.

N

[15] Anthony Edward Nelson. *Implementation of image processing algorithms on FPGA hardware*. Msc. thesis., Vanderbilt University, 2000.

[16] Jim Nichols and Sandeep Neema. Dynamically Reconfigurable Embedded Image Processing System. *Proceedings of the International Conference on Signal Processing Applications and Technology*, pages 1–5, 1999.

O

[17] Francisco Ortiz Zamora, Fernando Torres Medina, and Jesús Angulo López. Segmentación de imágenes aéreas mediante matemática en color y geodesia cromática. pages 1–7, 2002.

[18] Jorge Osio, Jose Rapallini, A A Quijano, and Jesús Ocampo. Implementación de un Algoritmo para procesamiento de imágenes en una FPGA. *Congreso de Microelectrónica Aplicada 2010*, pages 38–42, 2010.

Q

[19] Syed M Qasim, Shuja A Abbasi, and Bandar A Almashary. An overview of advanced FPGA architectures for optimized hardware realization of computation intensive algorithms. *2009 International Multimedia, Signal Processing and Communication Technologies*, pages 300–303, 2009.

R

[20] Carlos Rodríguez Cruz, Raziel Rivero Flores, Alejandro Castillo Atoche, and Javier Vázquez Castillo. Procesamiento de Imágenes con Xilinx System generator. *Primer Congreso Internacional de Sistemas Computacionales y electrónicos*, pages 1–8, 2006.

S

[21] T Saidani, D Dia, W Elhamzi, M Atri, and R Tourki. Hardware Co-simulation For Video Processing Using Xilinx System Generator. *Proceedings of the World Congress on Engineering 2009*, I:3–7, 2009.

[22] Alba M Sánchez G., Ricardo Alvarez G., and Sully Sánchez G. Architecture for filtering images using Xilinx system generator. In *International Journal of Mathematics and Computer in Simulation*, volume 1, pages 101–107, 2007.

[23] Alberto Sangiovanni-Vincentelli. The tides of EDA. *Design & Test of Computers, IEEE*, 20(6):59–75, 2005.

[24] Kirti Sikri Desai. EDA Innovation through Merger and Acquisitions. at Web Page: http://www10.edacafe.com, 2006.

[25] P Soille. *Morphological Image Analysis: Principles and Applications*, pages 1–4, 6–8, 17–26, 63–70, 80, 105–109, 267–268, 319. Springer-Verlag, 2nd edition, 1999.

[26] Gabor Szedo. Two-dimensional rank-order filter by using max-min sorting network. *Xilinx Application Notes*, pages 1–17, 2006.

T

[27] Ana Toledo Moreo, Juan Suardíaz Muro, and Sergio Cuenca Asensi. Entorno de codiseño para sistemas heterogéneos de procesamiento de imagen. pages 1–7, 2003.

[28] Ana Toledo Moreo, Cristina Vicente-Chicote, Juan Suardíaz Muro, and Sergio Cuenca Asensi. Xilinx System Generator Based HW Components for Rapid Prototyping of Computer Vision SW/HW Systems. *Pattern Recognition and Image Analysis*, pages 667–674, 2005.

[29] Wuilian J Torres and Roger J Bello. Procesamiento de imágenes a color utilizando morfología matemática. pages 1–5, 2002.

[30] Robert Turney. Two-Dimensional Linear Filtering. *Xilinx Application Notes*, 933:1–8, 2007.

V

[31] Luc Vincent and Edward R Dougherty. *Morphological Segmentation for Textures and Particles*, pages 43–102. 1994.

W

[32] Marek Wnuk. Remarks on Hardware Implementation of Image Processing Algorithms. *International Journal of Applied Mathematics and Computer Science*, 18(1):105–110, 2008.

X

[33] Xilinx. The MathWorks Design Tools and Service . at Web Page: http://www.xilinx.com, 2009.

[34] Xilinx. Spartan-3A DSP FPGA Family Data Sheet. Data Sheet, 2010.

[35] Xilinx. Spartan-3A FPGA Family Data Sheet. Data Sheet, 2010.

[36] Xilinx. *Xilinx System Generator for DSP Getting Started Guide*, chapter 1, 2, 4, pages 16, 19, 44. 2010.

[37] Xilinx. *Xilinx System Generator for DSP Reference Guide*, pages 44–47, 51–54, 75, 86–88, 116–119, 207–210, 223, 229, 265, 319, 329–334. Xilinx, 2010.

Anexos

Anexo 1: Componentes de la biblioteca XSGImgLib

La biblioteca de procesado de imágenes XSGImgLib presenta un total de 19 bloques de procesado lineal, 5 bloques de procesado no lineal, 8 bloques de procesado morfológico y 4 bloques de lógica de control.

Paleta de procesado lineal

La paleta de procesado lineal (Tabla 6.1) presenta 4 bloques de convolución genéricos, para ventanas de 3×3 y 5×5. Estos bloques mezclan la precisión de las operaciones (total precisión o normalizada) y la simetría del kernel (no simétrico o simétrico). Además, permiten la configuración del pipeline de las operaciones para realizar una optimización en consumo de recursos o en frecuencia de ejecución. Los bloques específicos de esta paleta implementan filtros ya definidos y optimizados para los kernels de convolución. Éstos se dividen en 8 filtros para ventanas de 3×3 y 3 filtros para 5×5.

TABLA 6.1: Paleta de procesado lineal de la biblioteca XSGImgLib.

Bloques	**Arquitectura**		**Operación**	**Parám.**	
	3×3	5×5		A/V	Norm
Conv. genérica no simétrica total precisión	sí	sí	*Sobel X*, *Y*, *Prewitt X*, *Y*, Externo	sí	no
Conv. genérica no simétrica normalizada	sí	sí	*Sobel X*, *Y*, *Prewitt X*, *Y*, Externo	sí	sí

Bloques	Arquitectura		Operación	Parám.	
	3×3	5×5		A/V	Norm
Conv. genérica simétrica total presición	sí	sí	*Edge*, *Laplace*, *Sobel XY*, *Prewitt XY*, *Sharpen*, *Blur*, *Smooth*, *Gaussian*, Externo	sí	no
Conv. genérica simétrica normalizada	sí	sí	*Edge*, *Laplace*, *Sobel XY*, *Prewitt XY*, *Sharpen*, *Blur*, *Smooth*, *Gaussian*, Externo	sí	sí
Filtro *Sobel X*, Filtro *Sobel Y*, Filtro *Sobel XY*	sí	no	*Sobel X*, *Sobel Y*, *Sobel XY* optimizados	no	no
Filtro *Prewitt X*, Filtro *Prewitt Y*, Filtro *Prewitt XY*	sí	no	*Prewitt X*, *Prewitt Y*, *Prewitt XY* optimizados	no	no
Filtro *Edge*	sí	no	*Edge* optimizado	no	no
Filtro *Laplace*	sí	no	*Laplace* optimizado	no	no
Filtro *Sharpen*	sí	no	*Sharpen* optimizado	no	no
Filtro *Blur*	no	sí	*Blur* optimizado	no	no
Filtro *Smooth*	no	sí	*Smooth* optimizado	no	no
Filtro *Gaussian*	no	sí	*Gaussian* optimizado	no	no

Conv.: Convolución.

A/V: Optimización en área o velocidad.

Norm: Selección de la cantidad de bits del punto decimal de las operaciones y los coeficientes del kernel.

Paleta de procesado no lineal

La paleta de procesado no lineal (Tabla 6.2) presenta 3 filtros basados en algoritmos de ordenamiento y un umbralizador. Los filtros de ordenamiento genéricos (para vecindades de 3×3 y 5×5) permiten la selección de la posición deseada. El filtro específico de mediana

(3×3) devuelve la salida del píxel en la posición central. El bloque de umbralización presenta como parámetros el nivel de umbral requerido en el procesado de la imagen.

TABLA 6.2: Paleta de procesado no lineal de la biblioteca XSGImgLib.

Bloques	**Arquitectura**		**Operación**	**Parám.**	
	3×3	5×5		Rank	THRS
Filtro de ordenamiento genérico	sí	sí	Mediana, Máximo, Mínimo, posición	sí	no
Filtro de Mediana 2D	sí	no	Mediana optimizada	no	no
Umbralizador	-	-	*Threshold*	no	sí

Rank: Selección de la posición.

THRS:Selección del valor de umbral (*threshold*).

Paleta de procesado morfológico

El procesado morfológico de la biblioteca (Tabla 6.3) está compuesto por 4 bloques para arquitecturas de 3×3 y 5×5. Los operadores de erosión y dilatación permiten la elección de elemento estructural, mientras que los bloques de apertura y cierre agregan además la cantidad de columnas de la imagen.

TABLA 6.3: Paleta de procesado morfológico de la biblioteca XSGImgLib.

Bloques	**Arquitectura**		**Operación**	**Parám.**	
	3×3	5×5		SE	Col
Operador erosión	sí	sí	Erosión optimizada	sí	no
Operador dilatación	sí	sí	Dilatación optimizada	sí	no
Operador apertura	sí	sí	Apertura optimizada	sí	sí
Operador cierre	sí	sí	Cierre optimizado	sí	sí

SE: Selección del elemento estructural.

Col: Selección de la cantidad de columnas.

Paleta de control de la biblioteca XSGImgLib

La paleta de control de la biblioteca XSGImgLib (Tabla 6.4) cuenta con arquitecturas para vecindades de 3×3 y 5×5 para los bloques almacenadores de línea y de registros. Éstos son utilizados en la paralelización de la información.

TABLA 6.4: Paleta de control de la biblioteca XSGImgLib.

Bloques	**Arquitectura**		**Operación**	**Parám.**
	3×3	5×5		Col
Line Buffers	sí	sí	Permite la paralelización de los datos	sí
Registers	sí	sí	Permite la paralelización de los datos	no

Col: Selección de la cantidad de columnas.

Anexo 2: Núcleos de convolución de procesado lineal

Los núcleos de convolución desarrollados en la biblioteca XSGImgLib se dividen en tres grupos principales de acuerdo a la acción sobre la imagen de entrada. Los núcleos de detección de bordes delimitan los contornos de los objetos en las imágenes, los de suavizado se encargan de eliminar el ruido y las discontinuidades, mientras que los de realce mejoran los bordes y otras regiones en las mismas. Estos núcleos se presentan a continuación.

TABLA 6.5: Núcleos de detección de bordes.

-1	-1	-1
-1	8	-1
-1	-1	-1

Edge

-1	0	1
-2	0	2
-1	0	1

Sobel X

-1	-2	-1
0	0	0
1	2	1

Sobel Y

-1	-1	0
-1	0	1
0	1	1

Sobel X-Y

0	-1	0
-1	4	-1
0	-1	0

Laplace

-1	0	1
-1	0	1
-1	0	1

Prewitt X

-1	-1	-1
0	0	0
1	1	1

Prewitt Y

-2	-1	0
-1	0	1
0	1	2

Prewitt X-Y

TABLA 6.6: Núcleos de suavizado.

1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1

Blur

1	1	1	1	1
1	5	5	5	1
1	5	44	5	1
1	5	5	5	1
1	1	1	1	1

Smooth

1	1	2	1	1
1	2	4	2	1
2	4	8	4	2
1	2	4	2	1
1	1	2	1	1

Gaussian

TABLA 6.7: Núcleo de realce.

-2	-2	-2
-2	32	-2
-2	-2	-2

Sharpen

Anexo 3: Bloques básicos utilizados

Los bloques básicos brindados por System Generator permiten el desarrollo de sistemas de procesado más complejos, utilizando el flujo de diseño basado en modelos. El uso racional y parametrizado de estos bloques permite el ahorro de recursos y el aumento de la frecuencia, implementando sistemas competitivos con los desarrollados utilizando las herramientas de diseño convencionales. Los bloques básicos empleados para el desarrollo de la biblioteca de procesado XSGImgLib se describen a continuación.

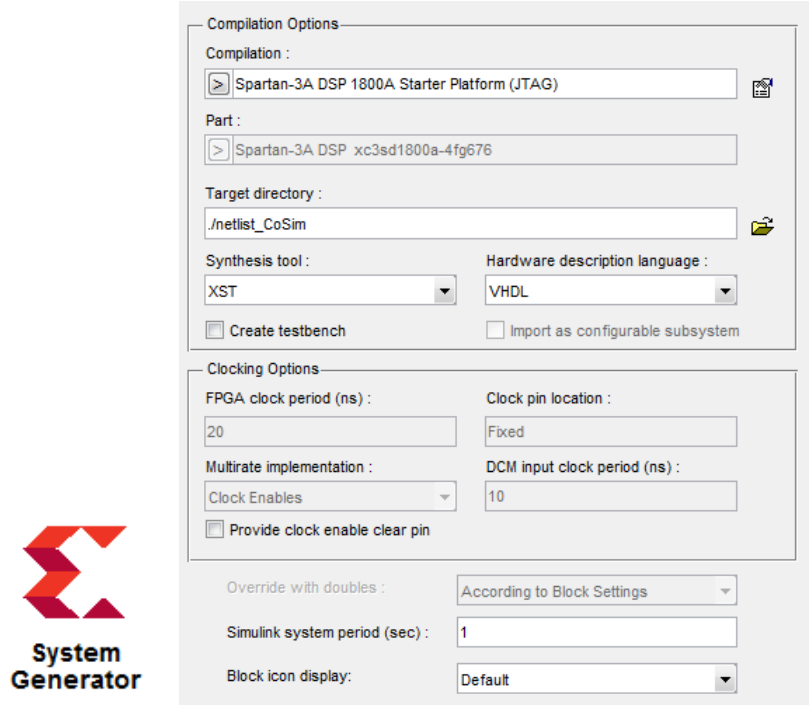
Bloque de System Generator

El bloque de System Generator, (*System Generator Token*) mostrado en la Figura 6.1, permite el control del sistema y los parámetros de simulación. Además, éste invoca al generador de código cuando se desea realizar la implementación del sistema en hardware. Todo diseño en Simulink que utilice los bloque básicos de Xilinx debe contener un bloque de System Generator.

Los parámetros de este bloque en la ventana de configuración (Figura 6.1b) definen las opciones de compilación (*"Compilation Options"*), las opciones de temporización (*"Clocking Options"*) y las opciones para la simulación (*"Other Options"*) del diseño. En las opciones de compilación (campo *"Compilation"*) se especifica el tipo de resultado que debe ser obtenido al invocar al generador de código. Entre ellos se encuentran: la generación de un fichero de configuración (*Bitstream*), la obtención de un modelo para la co-simulación hardware y el llamado de la herramienta de exportación del modelo a EDK, entra otras. El campo *"Part"* especifica el FPGA a utilizar. El directorio de destino se introduce en el campo *"Target directory"*. En esta dirección, System Generator escribe los resultados de la compilación del diseño. Las herramientas de síntesis y el lenguaje de descripción de hardware se seleccionan en los campos *"Synthesis tool"* y *"Hardware description language"* respectivamente. Entre las herramientas de síntesis disponibles se encuentran XST, Synplify y Synplify Pro, mientras que entre los lenguajes a utilizar se pueden citar VHDL y Verilog.

Los parámetros de temporización permiten seleccionar el período de reloj del FPGA en nanosegundos (*"FPGA clock period(ηs)"*), la localización del pin de reloj (*"Clock pin location"*), la implementación del sistema utilizando multifrecuencia (*"Multirate implementation"*), entre otras. Estos parámetros son enviados al FPGA mediante el archivo de restricciones.

Entre las demás opciones del bloque están: el período del sistema en la simulación (*"Simulink system period(sec)"*), y el icono a mostrar en los bloques (*"Block icon display"*). De acuerdo a este parámetro los bloques del diseño muestran información acerca de la frecuencia de muestreo, los nombres de los puertos, las etapas de pipeline, entre otras.



(a) Diagrama. **(b)** Propiedades.

Figura 6.1: Bloque de System Generator [37].

Puerta de entrada

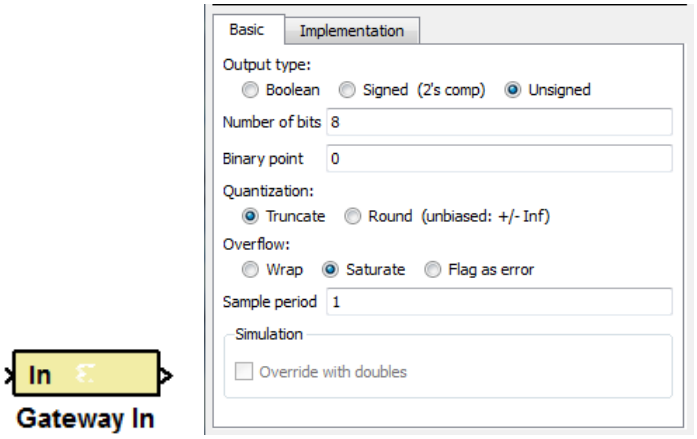
La puerta de entrada (*Gateway In*) de la Figura 6.2 es el bloque de acceso de información al diseño en Simulink. Éste convierte el tipo de datos de MATLAB en valores de punto fijo para XSG. En la conversión de datos, el bloque de entrada utiliza las opciones de limitación de los valores de entrada. Estos métodos no ocupan recursos en hardware ya que se ejecutan en software antes de la simulación del diseño.

Entre las funciones que ejecuta la puerta de entrada se encuentran: la conversión de datos desde Simulink, la definición del nivel superior de entrada de datos en el diseño en

HDL generado, la definición del banco de pruebas para la comprobación del mismo y la denominación de las señales en la entidad.

Los parámetros de la puerta de entrada (Figura 6.2b) permiten la configuración de las restricciones de tiempo de los almacenadores de entrada/salida (*"IOB Timing Constraint"*), el posicionado de los mismos (*"Specify IOB Location Constraints"*) y las especificaciones de las celdas lógicas de entrada/salida (*"IOB Pad Locations"*).

Las restricciones temporales de los almacenadores de entrada/salida permite configurar la razón de transferencia de datos de los IOB determinada por el período de sistema, mientras que el posicionado establece el lugar de ubicación de los IOB en el dispositivo físico, así como los terminales a utilizar por los mismos.



(a) Diagrama. **(b)** Propiedades.

Figura 6.2: Bloque de puerta de entrada [37].

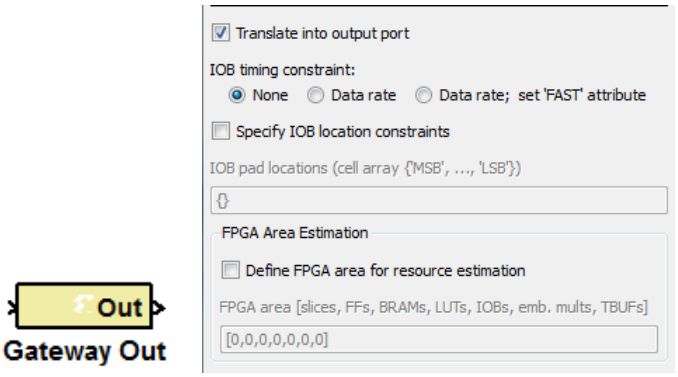
Puerta de salida

La puerta de salida (*Gateway Out*), mostrada en la Figura 6.3, es la vía mediante la cual se entregan los datos procesados en el diseño XSG al espacio de trabajo de Simulink. Este bloque define los puertos de salida del diseño en HDL para el dispositivo físico, o simplemente puede utilizarse como un punto de prueba para la representación hardware.

La puerta de salida realiza la conversión de la representación numérica de XSG a doble precisión de Simulink, define los bancos de pruebas durante la generación de código HDL y denomina las señales de salida para la entidad.

Entre los parámetros principales (Figura 6.3b) se encuentran la habilitación como puerto de salida en el hardware (*"Translate into Output Port"*), las restricciones de tiempo de los almacenadores de entrada/salida (*"IOB Timing Constraint"*), el posicionado de los mismos (*"Specify IOB Location Constraints"*) y las especificaciones de las celdas lógicas de entrada/salida (*"IOB Pad Locations"*).

La habilitación de la salida como un terminal permite la selección de un IOB en el dispositivo físico, mientras que si esta opción no está activada la misma es utilizada como un punto de prueba durante la simulación en software. Los demás parámetros han sido descritos en el bloque anterior.



(a) Diagrama. **(b)** Propiedades.

Figura 6.3: Bloque de puerta de entrada [37].

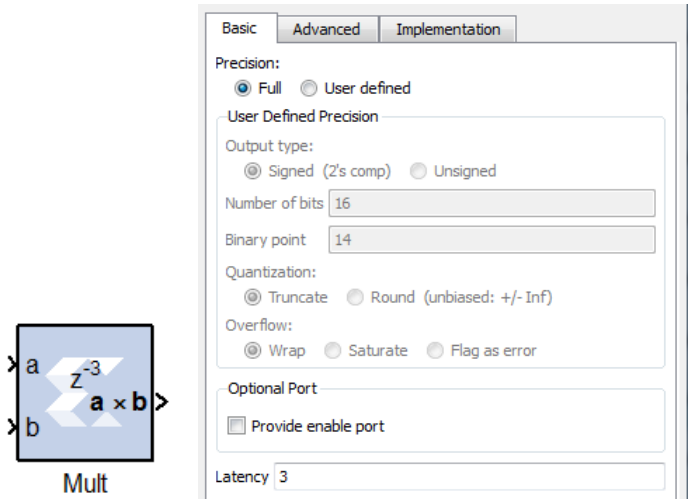
Bloque de multiplicación

El bloque de multiplicación de Xilinx (*Mult*) de la Figura 6.4 implementa un multiplicador de dos entradas, devolviendo el resultado del cálculo en su salida.

Los parámetros de este bloque (Figura 6.4b) se dividen en dos pestañas principales. En la pestaña básica se encuentran parámetros como la latencia (*"Latency"*), la cual define el número de períodos de reloj por el cual se retarda la salida del bloque y está relacionado con la cantidad de ciclos que demora el cálculo de la operación. Además, pueden configurarse en esta pestaña los parámetros de definición de la salida (*"User Defined Precision"*), entre ellos: el tipo de representación numérica, la cantidad de bits de salida, la cantidad de bits para la parte decimal (debe ser menor o igual que el parámetro anterior) y el método de

ajuste para la limitación de la cantidad de bits de salida. Estos parámetros son comunes en muchos bloques básicos de XSG.

En la pestaña de implementación se especifican los parámetros de la misma en el dispositivo. En ésta, es posible seleccionar el uso de la descripción por comportamiento en HDL (*"Use behavioral HDL"*). Esta opción optimiza la herramienta de síntesis para el uso de la menor área del dispositivo. Entre otros parámetros de esta pestaña se encuentran: la optimización del diseño (*"Optimize for Speed | Area"*), el uso de multiplicadores dedicados para los bloques de multiplicación (*"Use embedded multipliers"*), y el análisis para el pipeline óptimo (*"Test for optimium pipelining"*). Este último realiza la simulación del sistema buscando el valor de latencia para alcanzar la frecuencia de trabajo máxima.



(a) Diagrama. **(b)** Propiedades.

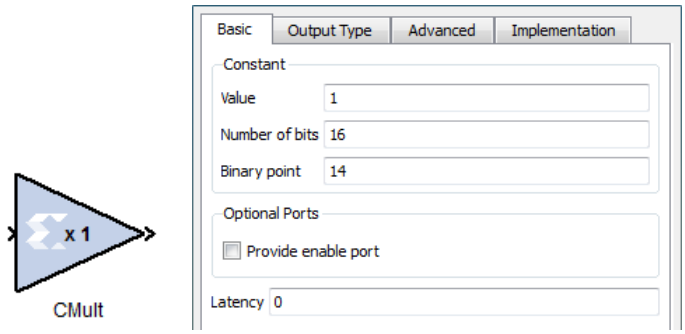
Figura 6.4: Bloque de multiplicación [37].

Bloque de multiplicación por una constante

El bloque de multiplicación por una constante (*CMult*), mostrado en la Figura 6.5, implementa un operador de ganancia por un valor configurado, con salida igual a la multiplicación de la entrada por la constante.

El bloque cuenta con tres pestañas de parámetros (Figura 6.5b). En la pestaña básica se selecciona el valor de la constante (*"Constant value"*), así como la cantidad de bits para la representación de la misma (*"Constant Number of bits"*) y su parte decimal (*"Constant*

Binary point"). La pestaña de parámetros de salida configura el tipo de representación numérica a utilizar por el operador, explicada anteriormente, mientras que la pestaña de la implementación configura el uso de descripción por comportamiento, la utilización de bloques de memoria distribuidas o memorias dedicadas (*"Implement using"*), además del análisis para el pipeline óptimo.



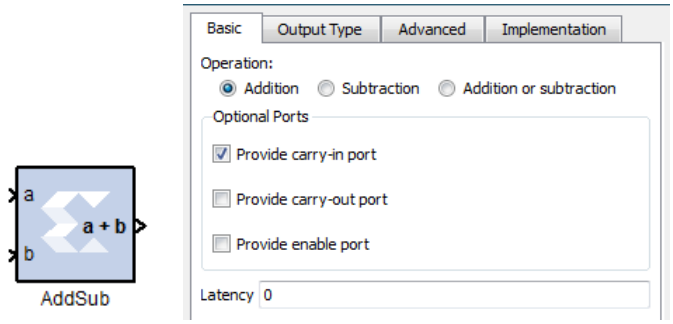
(a) Diagrama. **(b)** Propiedades.

Figura 6.5: Bloque de multiplicación por una constante [37].

Bloque de suma/resta

El bloque de suma/resta de System Generator (*AddSub*) de la Figura 6.6 desarrolla un bloque de suma o resta de dos entradas y una salida.

El bloque presenta cuatro pestañas de configuración (Figura 6.6b). La pestaña básica presenta la selección del tipo de operación entre suma, resta o ambas (*"Operation"*), además de la habilitación de un bit de acarreo de entrada (*"Provide carry-in Port"*) y/o salida (*"Provide carry-out Port"*) en el bloque. La pestaña de implementación posibilita la elección de la descripción por comportamiento HDL, la implementación utilizando recursos dedicados, y la optimización del rendimiento para el máximo pipeline (*"Pipeline for maximum performance"*). Este parámetro optimiza la frecuencia de ejecución mediante el pipeline interno de la operación. Las restantes pestañas de configuración presentan parámetros genéricos de los bloques ya explicados, entre ellos la representación numérica de la salida y los métodos de ajuste de la misma.



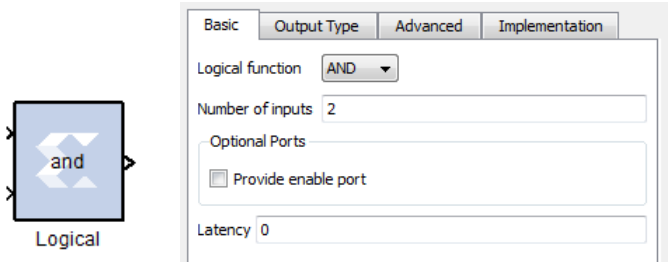
(a) Diagrama. **(b)** Propiedades.

Figura 6.6: Bloque de suma/resta [37].

Bloque de operaciones lógicas

El bloque de operaciones lógicas de Xilinx (*Logical*) de la Figura 6.7 realiza diferentes cálculos bit a bit entre 2, 3 ó 4 números de punto fijo. El resultado de estas operaciones es devuelto mediante la salida del bloque. Éste es implementado mediante VHDL sintetizable, permitiendo la unión de varias compuertas lógicas en cascada para optimizar la ejecución de la operación final.

El bloque presenta cuatro pestañas de configuración (Figura 6.7b). La pestaña básica selecciona la función lógica entre los operadores AND, $NAND$, OR, NOR, XOR y $XNOR$ (*"Logical function"*) y especifica la cantidad de entradas entre 1 y 1024 (*"Number of inputs"*). La pestaña de configuración de la salida especifica si ésta debe ajustar automáticamente el punto decimal (*"Align binary point"*), además de otros parámetros ya explicados en bloques anteriores.

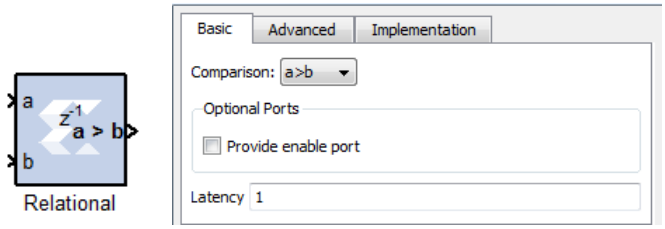


(a) Diagrama. **(b)** Propiedades.

Figura 6.7: Bloque de operaciones lógicas [37].

Bloque de operaciones relacionales

El bloque relacional de System Generator (*Relational*) de la Figura 6.8 implementa un comparador con varias funciones ya definidas. La salida del mismo es un bit que representa el resultado de la comparación. Este bloque presenta tres pestañas de configuración (Figura 6.8b) con varios parámetros anteriormente explicados. La pestaña básica agrega la selección de la operación a realizar (*"Comparison"*).

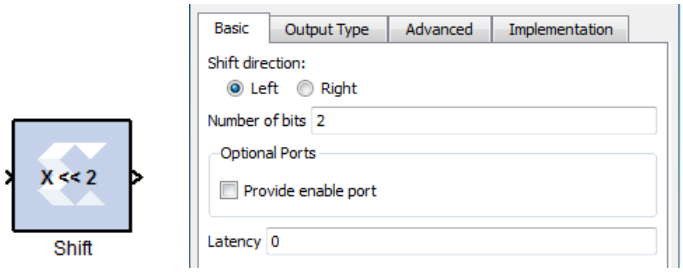


(a) Diagrama. **(b)** Propiedades.

Figura 6.8: Bloque de operaciones relacionales [37].

Bloque de rotaciones

El bloque básico de rotación (*Shift*), mostrado en la Figura 6.9, ejecuta la rotación del valor en la entrada hacia la derecha o a la izquierda, implementando así multiplicaciones y divisiones en potencias de 2. El resultado presenta el mismo punto fijo que la entrada. En la pestaña de configuración básica (Figura 6.9b), el bloque presenta la selección de la dirección de rotación (*"Shift direction"*), multiplicando por la potencia de 2 si la rotación es a la izquierda, mientras que realiza una división rotando a la derecha. Además, la cantidad de bits a rotar (*"Number of bits"*) puede ser especificada en la misma. Los demás parámetros de este bloque fueron explicados anteriormente.

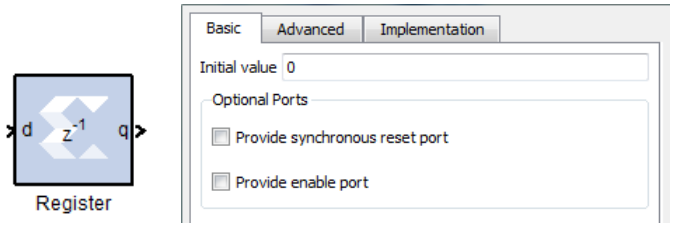


(a) Diagrama. **(b)** Propiedades.

Figura 6.9: Bloque de rotaciones [37].

Bloque de almacenamiento

El bloque de registro (*Register*), mostrado en la Figura 6.10, modela un almacenador tipo D con una latencia de un período de muestreo. Este bloque mantiene el valor de la entrada durante un ciclo de reloj devolviéndolo a la salida luego de éste. Los parámetros del bloque (Figura 6.10b) presentan configuraciones anteriormente descritas, agregando la opción de precisar el valor inicial (*"Initial value"*) del mismo. Además provee entradas adicionales (*"Optional Ports"*) como $reset$ y $enable$.



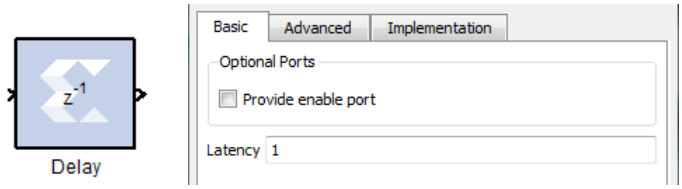
(a) Diagrama. **(b)** Propiedades.

Figura 6.10: Bloque de almacenamiento [37].

Bloque de retardo

El bloque de retardo de la señal (*Delay*) de la Figura 6.11 demora la señal de entrada L períodos de muestreo. El mismo se diferencia del anterior en que no presenta configuraciones iniciales ni señal de $reset$. Además, la pestaña básica (Figura 6.11b) permite la selección del valor de retardo (*"Latency"*), mientras que la elección de la descripción por comportamiento

utilizando HDL (*"Implement using behavioral HDL"*) puede ser configurada en la pestaña de implementación. Los restantes parámetros han sido descritos en bloques anteriores.

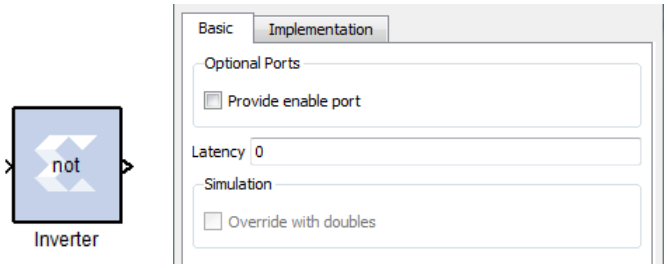


(a) Diagrama. **(b)** Propiedades.

Figura 6.11: Bloque de retardo [37].

Bloque de negado

El bloque de inversión o negado (*Inverter*) de la Figura 6.12 calcula el complemento lógico bit a bit de un valor con punto fijo en la entrada. El mismo utiliza VHDL sintetizable para su implementación. Los parámetros de este bloque (Figura 6.12b) han sido explicados anteriormente.



(a) Diagrama. **(b)** Propiedades.

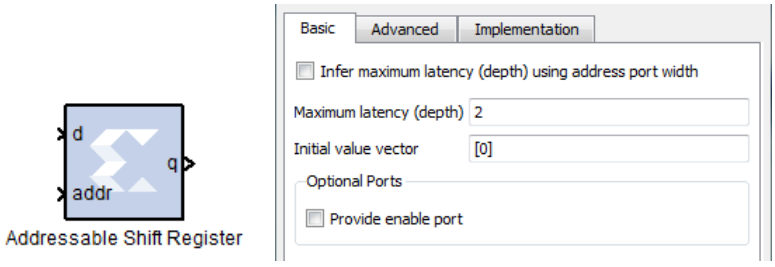
Figura 6.12: Bloque de negado [37].

Bloque de registro de desplazamiento

El bloque de registro de desplazamiento direccionable (*Addressable Shift Register*), mostrado en la Figura 6.13, implementa una cola FIFO de valores, desplazándolos en cada ciclo de reloj, y permitiendo la habilitación a la salida de cualquier etapa de ésta. Internamente el bloque está compuesto por una serie de registros cuyas salidas son enviadas a un mul-

tiplexor, el cual habilita a la salida del bloque el valor en la posición especificada por la entrada de selección.

El bloque presenta tres pestañas de configuración (Figura 6.13b) de las cuales muchos parámetros han sido explicados en bloques anteriores. La pestaña básica presenta un parámetro para autoajustar la cantidad de etapas de almacenamiento (*"Infer maximum latency (depth) using address port width"*) según el valor en la entrada de dirección. En caso de no seleccionar este parámetro la cantidad de etapas puede ser configurada en el campo *"Maximum latency (depth)"*. Este valor no puede ser menor que 2 mi mayor que 1024 según los parámetros del bloque. Además, el mismo permite especificar la etapa inicial del registro (*"Initial value vector"*), ignorando las etapas anteriores a ésta.



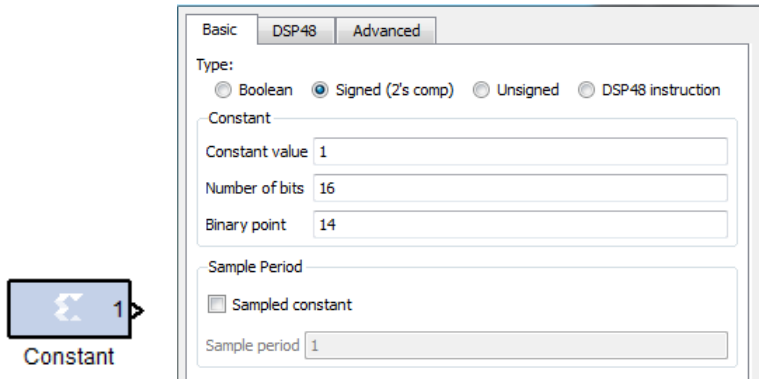
(a) Diagrama. **(b)** Propiedades.

Figura 6.13: Bloque de registro de desplazamiento [37].

Bloque de constante

El bloque de constante (*Constant*), mostrado en la Figura 6.14, almacena un valor constante que puede ser un número de punto fijo, un valor booleano, o una instrucción DSP48. Este último es utilizado para obtener el control de la secuencia de instrucciones para los bloques DSP48.

Este bloque presenta tres pestañas de configuración (Figura 6.14b). La pestaña básica permite la selección del tipo de constante (*"Type"*) entre los ya mencionados, el valor de la constante (*"Constant Value"*), así como el período de muestreo inherente a la misma (*"Sampled Constant"*). La pestaña DSP48 se activa al seleccionar el tipo *"Instrucción DSP48"* permitiendo el trabajo con estos bloques específicos.

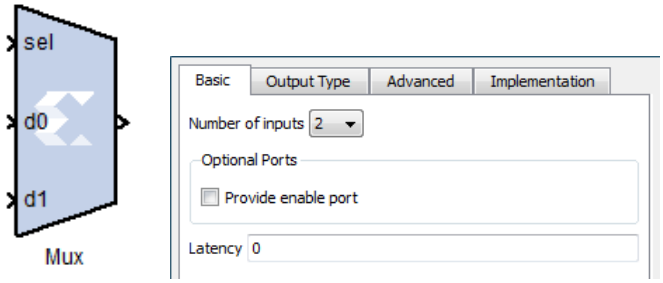


(a) Diagrama. **(b)** Propiedades.

Figura 6.14: Bloque de constante [37].

Bloque de multiplexación

El bloque de multiplexado (*Mux*) de la Figura 6.15 implementa un multiplexor con una entrada de selección, n canales de entrada al bloque y una salida. La configuración del mismo (Figura 6.15b) presenta los parámetros básicos anteriormente expuestos. Además permite la selección de la cantidad de canales de entrada al multiplexor entre 2 y 1024.



(a) Diagrama. **(b)** Propiedades.

Figura 6.15: Bloque de multiplexación [37].

Anexo 4: Elementos estructurales

Los elementos estructurales son parte de la configuración de los bloques de procesado morfológico. El cambio de estos valores permite obtener varios resultados utilizando el mismo bloque de procesamiento, incidiendo en la cantidad de píxeles en la operación. A continuación se muestran configuraciones básicas de los elementos estructurales para ventanas de 3×3 y 5×5.

TABLA 6.8: Elementos estructurales básicos para ventanas de 3×3.

1	1	1	0	1	0	0	0	1	1	0	0	1	1	1	0	0	0
1	1	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1
Ones			*Cross*			*Right*			*Left*			*Up*			*Bottom*		

TABLA 6.9: Elementos estructurales básicos para ventanas de 5×5.

1	1	1	1	1	0	1	1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	1	1	0	1	1	1	0	0	0	1	0	0
Ones					*Disc*					*Diamond*				
0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
Cross					*Right*					*Left*				

1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1
Up					*Bottom*				

Anexo 5: Latencia de los bloques de procesado

Los bloques de procesado de la biblioteca XSGImgLib presentan una latencia debido a las etapas internas de las operaciones. Esta latencia (τ_{OP}) se muestra en la Tabla 6.10 para cada uno de ellos.

TABLA 6.10: Latencia de los bloques de procesado.

Bloque de procesado	**Optimización**	τ_{op}	**Bloque de procesado**	τ_{op}
Convolución genérico	Área	6	Filtro *Laplace*	2
3×3	Velocidad	8	Filtro *Sharpen*	2
Convolución genérico	Área	7	Filtro *Blur*	4
5×5	Velocidad	9	Filtro *Smooth*	4
Filtro *Sobel X*	-	2	Filtro *Gaussian*	4
Filtro *Sobel Y*	-	2	Filtro de Ordenamiento genérico	$2rw + 1$
Filtro *Sobel X-Y*	-	2	Filtro de Mediana	2
Filtro *Prewitt X*	-	2	Operador Dilatación	$2rw$
Filtro *Prewitt Y*	-	2	Operador Erosión	$2rw$
Filtro *Prewitt X-Y*	-	2	Operador Apertura	$(2rw)(2 + n)$
Filtro *Edge*	-	2	Operador Cierre	$(2rw)(2 + n)$

τ_{op}: Latencia de la operación en ciclos de reloj.
rw: Radio de la ventana.
n: Número de columnas.

Anexo 6: Consumo de recursos

TABLA 6.11: Consumo de recursos: convolución 3×3 no simétrica optimizada en área o velocidad en una *Spartan-3A SK 700a*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	141	2.39 %	316	5.37 %
Mult 18×18	9	45.00 %	9	45.00 %
Frecuencia	152.555 MHz		180.310 MHz	

TABLA 6.12: Consumo de recursos: convolución 3×3 no simétrica optimizada en área o velocidad en una *Spartan-3A DSP 1800*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	141	0.85 %	215	1.29 %
DSP48A	9	10.71 %	9	10.71 %
Frecuencia	136.054 MHz		182.216 MHz	

TABLA 6.13: Consumo de recursos: convolución 5×5 no simétrica optimizada en área o velocidad en una *Spartan-3A DSP 1800*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	343	2.06 %	543	3.26 %
DSP48A	25	29.76 %	25	29.76 %
Frecuencia	136.054 MHz		148.214 MHz	

TABLA 6.14: Consumo de recursos: convolución 3×3 simétrica optimizada en área o velocidad en una *Spartan-3A SK 700a*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	125	2.12 %	245	4.16 %
Mult 18×18	6	30.00 %	6	30.00 %
Frecuencia	164.447 MHz		186.741 MHz	

TABLA 6.15: Consumo de recursos: convolución 5×5 simétrica optimizada en área o velocidad en una *Spartan-3A SK 700a*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	303	5.15 %	608	10.33 %
Mult 18×18	15	75.00 %	15	75.00 %
Frecuencia	107.527 MHz		177.904 MHz	

TABLA 6.16: Consumo de recursos: convolución 3×3 simétrica optimizada en área o velocidad en una *Spartan-3A DSP 1800*.

Recursos	**Área**		**Velocidad**	
	Utilizados	Porciento	Utilizados	Porciento
Slices	125	0.85 %	176	1.29 %
DSP48A	6	7.14 %	6	7.14 %
Frecuencia	136.054 MHz		181.389 MHz	

TABLA 6.17: Consumo de recursos: convolución 5×5 simétrica optimizada en área o velocidad en una *Spartan-3A DSP 1800*.

Recursos	Área		Velocidad	
	Utilizados	Porciento	Utilizados	Porciento
Slices	303	1.82 %	433	2.60 %
DSP48A	15	17.86 %	15	17.86 %
Frecuencia	136.054 MHz		158.604 MHz	

TABLA 6.18: Consumo de recursos: convolución 3×3 no simétrica, normalizada y optimizada en velocidad en una *Spartan-3A DSP 1800*.

Recursos	Punto decimal de 3 bits		Punto decimal de 6 bits		Punto decimal de 8 bits	
	Utilizados	Porciento	Utilizados	Porciento	Utilizados	Porciento
Slices	154	0.93 %	195	1.17 %	214	1.29 %
DSP48A	9	10.71 %	9	10.71 %	9	10.71 %
Frecuencia	183.419 MHz		180.050 MHz		191.755 MHz	
MSE	9.80		0.22		0.029	

TABLA 6.19: Consumo de recursos: convolución 5×5 no simétrica, normalizada y optimizada en velocidad en una *Spartan-3A DSP 1800*.

Recursos	Punto decimal de 3 bits		Punto decimal de 6 bits		Punto decimal de 8 bits	
	Utilizados	Porciento	Utilizados	Porciento	Utilizados	Porciento
Slices	402	2.42 %	507	3.05 %	558	3.35 %
DSP48A	25	29.76 %	25	29.76 %	25	29.76 %
Frecuencia	164.989 MHz		165.508 MHz		170.999 MHz	
MSE	0		0.22		0.029	

TABLA 6.20: Consumos de recursos: convolución XSG 5×5 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	381	5	5
	Porciento	6.47 %	25 %	25 %

TABLA 6.21: Consumos de recursos: convolución XSG 5×5 en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**	**BRAM**
Recursos	Total	16640	84	84
	Utilizados	381	5	5
	Porciento	2.29 %	5.95 %	5.95 %

TABLA 6.22: Consumos de recursos: bloque de ordenamiento genérico de 3×3 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	253	0	0
	Porciento	4.30 %	0 %	0 %

TABLA 6.23: Consumos de recursos: bloque de ordenamiento genérico de 5×5 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	1339	0	0
	Porciento	22.74 %	0 %	0 %

TABLA 6.24: Consumos de recursos: bloque de ordenamiento genérico de 3×3 en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**	**BRAM**
Recursos	Total	16640	84	84
	Utilizados	253	0	0
	Porciento	1.52 %	0 %	0 %

TABLA 6.25: Consumos de recursos: bloque de ordenamiento genérico de 5×5 en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**	**BRAM**
Recursos	Total	16640	84	84
	Utilizados	1339	0	0
	Porciento	8.05 %	0 %	0 %

TABLA 6.26: Consumos de recursos: filtro de mediana de 3×3 en una *Spartan-3A SK 700a*.

Spartan-3A SK 700a		**Slices**	**Mult** 18×18	**BRAM**
Recursos	Total	5888	20	20
	Utilizados	164	0	0
	Porciento	2.79 %	0 %	0 %

TABLA 6.27: Consumos de recursos: filtro de mediana de 3×3 en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**	**BRAM**
Recursos	Total	16640	84	84
	Utilizados	164	0	0
	Porciento	0.99 %	0 %	0 %

TABLA 6.28: Consumo de recursos: operadores morfológicos erosión o dilatación para diferentes ventanas en una *Spartan-3A DSP 1800*.

Recursos	3×3		5×5	
	Utilizados	Porciento	Utilizados	Porciento
Slices	8	0.05 %	27	0.16 %
Frecuencia	506.329 MHz		361.402 MHz	

TABLA 6.29: Consumo de recursos: operadores morfológicos apertura o cierre para diferentes ventanas e imágenes de 64 columnas en una *Spartan-3A DSP 1800*.

Recursos	3×3		5×5	
	Utilizados	Porciento	Utilizados	Porciento
Slices	15	0.09 %	47	0.28 %
Frecuencia	213.721 MHz		156.104 MHz	

TABLA 6.30: Consumo de recursos: almacenadores de línea para diferentes ventanas e imágenes de 64 columnas en una *Spartan-3A DSP 1800*.

Recursos	3×3		5×5	
	Utilizados	Porciento	Utilizados	Porciento
Slices	32	0.19 %	64	0.38 %
Frecuencia	180.408 MHz		182.682 MHz	

TABLA 6.31: Consumo de recursos: ejemplo de procesado con la biblioteca XSGImgLib en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**	**BRAM**
Recursos	Total	16640	84	84
	Utilizados	1200	15	0
	Porciento	7.21 %	17.26 %	0 %

TABLA 6.32: Consumo de recursos: etapa de filtrado de ruido en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	503	0
	Porciento	41.99 %	0 %

TABLA 6.33: Consumo de recursos: etapa de detección de contornos en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	324	0
	Porciento	27 %	0 %

TABLA 6.34: Consumo de recursos: etapa de binarización en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	2	0
	Porciento	0.17 %	0 %

TABLA 6.35: Consumo de recursos: etapa de suavizado en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	509	15
	Porciento	42.41 %	100 %

TABLA 6.36: Consumo de recursos: etapa de dilatación en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	41	0
	Porciento	3.42 %	0 %

TABLA 6.37: Consumo de recursos: etapa erosión en una *Spartan-3A DSP 1800*.

Spartan-3A DSP 1800		**Slices**	**DSP48A**
Recursos	Total utilizado	1200	15
	Utilizados	40	0
	Porciento	3.34 %	0 %

Anexo 7: Comparación de arquitecturas

TABLA 6.38: Consumo de recursos: convolución 3×3 no simétrica y simétrica en una *Spartan-3A SK 700a*.

Optimización	**Área**		**Velocidad**	
Recursos	**No Sim.**	**Sim.**	**No Sim.**	**Sim.**
Slices	141	125	316	245
Mult 18×18	9	6	9	6
Frecuencia	152.555 MHz	164.447 MHz	180.310 MHz	186.741 MHz

TABLA 6.39: Consumo de recursos: convolución 3×3 no simétrica y simétrica en una *Spartan-3A DSP 1800*.

Optimización	**Área**		**Velocidad**	
Recursos	**No Sim.**	**Sim.**	**No Sim.**	**Sim.**
Slices	141	125	215	176
DSP48A	9	6	9	6
Frecuencia	136.054 MHz	136.054 MHz	182.216 MHz	181.389 MHz

TABLA 6.40: Consumo de recursos: convolución 5×5 no simétrica y simétrica en una *Spartan-3A DSP 1800*.

Optimización	**Área**		**Velocidad**	
Recursos	**No Sim.**	**Sim.**	**No Sim.**	**Sim.**
Slices	343	303	543	433
DSP48A	25	15	25	15
Frecuencia	136.054 MHz	136.054 MHz	148.214 MHz	158.604 MHz

TABLA 6.41: Consumo de recursos: convolución 3×3 simétrica y filtro específico *Edge* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **Sim.**		**Filtro *Edge***
Optimización	**Área**	**Velocidad**	
Slices	125	176	88
DSP48A	6	6	0
Frecuencia	136.054 MHz	181.389 MHz	392.619 MHz

TABLA 6.42: Consumo de recursos: convolución 3×3 simétrica y filtro específico *Laplace* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **Sim.**		**Filtro *Laplace***
Optimización	**Área**	**Velocidad**	
Slices	125	176	49
DSP48A	6	6	0
Frecuencia	136.054 MHz	181.389 MHz	449.640 MHz

TABLA 6.43: Consumo de recursos: convolución 3×3 simétrica y filtro específico *Sharpen* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **Sim.**		**Filtro *Sharpen***
Optimización	**Área**	**Velocidad**	
Slices	125	176	94
DSP48A	6	6	0
Frecuencia	136.054 MHz	181.389 MHz	393.391 MHz

TABLA 6.44: Consumo de recursos: convolución 3×3 no simétrica y filtro específico *Sobel X* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **no Sim.**		**Filtro *Sobel X***
Optimización	**Área**	**Velocidad**	
Slices	141	215	78
DSP48A	9	9	0
Frecuencia	136.054 MHz	182.216 MHz	403.714 MHz

TABLA 6.45: Consumo de recursos: convolución 3×3 no simétrica y filtro específico *Sobel Y* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **no Sim.**		**Filtro *Sobel Y***
Optimización	**Área**	**Velocidad**	
Slices	141	215	61
DSP48A	9	9	0
Frecuencia	136.054 MHz	182.216 MHz	582.072 MHz

TABLA 6.46: Consumo de recursos: convolución 3×3 simétrica y filtro específico *Sobel X-Y* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **Sim.**		**Filtro *Sobel X-Y***
Optimización	**Área**	**Velocidad**	
Slices	125	176	65
DSP48A	6	6	0
BRAM	0	0	0
Frecuencia	136.054 MHz	181.389 MHz	361.795 MHz

TABLA 6.47: Consumo de recursos: convolución 3×3 no simétrica y filtro específico *Prewitt X* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **no Sim.**		**Filtro *Prewitt X***
Optimización	**Área**	**Velocidad**	
Slices	141	215	70
DSP48A	9	9	0
BRAM	0	0	0
Frecuencia	136.054 MHz	182.216 MHz	355.999 MHz

TABLA 6.48: Consumo de recursos: convolución 3×3 no simétrica y filtro específico *Prewitt Y* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **no Sim.**		**Filtro *Prewitt Y***
Optimización	**Área**	**Velocidad**	
Slices	141	215	61
DSP48A	9	9	0
BRAM	0	0	0
Frecuencia	136.054 MHz	182.216 MHz	292.227 MHz

TABLA 6.49: Consumo de recursos: convolución 3×3 simétrica y filtro específico *Prewitt X-Y* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 3×3 **Sim.**		**Filtro *Prewitt X-Y***
Optimización	**Área**	**Velocidad**	
Slices	125	176	65
DSP48A	6	6	0
BRAM	0	0	0
Frecuencia	136.054 MHz	181.389 MHz	406.174 MHz

TABLA 6.50: Consumo de recursos: convolución 5×5 simétrica y filtro específico *Blur* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 5×5 **Sim.**		**Filtro *Blur***
Optimización	**Área**	**Velocidad**	
Slices	303	433	166
DSP48A	15	15	0
BRAM	0	0	0
Frecuencia	136.054 MHz	158.604 MHz	332.447 MHz

TABLA 6.51: Consumo de recursos: convolución 5×5 simétrica y filtro específico *Smooth* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 5×5 **Sim.**		**Filtro *Smooth***
Optimización	**Área**	**Velocidad**	
Slices	303	433	318
DSP48A	15	15	0
BRAM	0	0	0
Frecuencia	136.054 MHz	158.604 MHz	210.970 MHz

TABLA 6.52: Consumo de recursos: convolución 5×5 simétrica y filtro específico *Gaussian* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen.** 5×5 **Sim.**		**Filtro *Gaussian***
Optimización	**Área**	**Velocidad**	
Slices	303	433	215
DSP48A	15	15	0
BRAM	0	0	0
Frecuencia	136.054 MHz	158.604 MHz	251.067 MHz

TABLA 6.53: Consumo de recursos: convolución XSG 5×5 y filtro específico *Smooth* en una *Spartan-3A DSP 1800*.

Bloque	**Conv. Gen. XSG** 5×5	**Conv. Gen.** 5×5 **no Sim.**		**Filtro *Smooth***
Optimización		**Área**	**Velocidad**	
Slices	381	343	543	318
DSP48A	5	25	25	0
BRAM	5	0	0	0
Frecuencia	27.211 MHz	136.054 MHz	148.214 MHz	210.970 MHz

TABLA 6.54: Consumo de recursos: filtro de mediana de 3×3 y bloque de ordenamiento genérico de 3×3 en una *Spartan-3A DSP 1800*.

Recursos	**Filtro Mediana** 3×3	**Rank Sorting** 3×3
Slices	164	253
Frecuencia	293.341 MHz	71.083 MHz

TABLA 6.55: Consumo de recursos: sistema de procesado con XSG y con VHDL de 3×3 en una *Spartan-3A DSP 1800*.

Recursos	**Sistema con filtro de mediana**	**Sistema con VHDL de** 3×3
Slices	1489	534
BRAM	0	2
Frecuencia	77.340 MHz	106.202 MHz

TABLA 6.56: Consumo de recursos: sistema de procesado con XSG y con VHDL de 5×5 en una *Spartan-3A DSP 1800*.

Recursos	**Sistema con bloque de ordenamiento**	**Sistema con VHDL de** 5×5
Slices	5193	2240
BRAM	0	4
Frecuencia	50.533 MHz	101.092 MHz

TABLA 6.57: Consumo de recursos: almacenadores de línea y *Virtex Line Buffer* para imágenes de 64 columnas en una *Spartan-3A DSP 1800*.

Bloque	**Almacenadores de línea**		***Virtex Line Buffer***	
Recursos	3×3	5×5	3×3	5×5
Slices	32	64	14	28
BRAM	0	0	2	4
Frecuencia	180.408 MHz	182.682 MHz	183.756 MHz	122.205 MHz

TABLA 6.58: Consumo de recursos: almacenadores de línea y *Virtex2 Line Buffer* para imágenes de 64 columnas en una *Spartan-3A DSP 1800*.

Bloque	**Almacenadores de línea**		***Virtex2 Line Buffer***	
Recursos	3×3	5×5	3×3	5×5
Slices	32	64	8	16
BRAM	0	0	2	4
Frecuencia	180.408 MHz	182.682 MHz	215.796 MHz	191.608 MHz

Nomenclatura

ASIC	Application Specific Integated Circuit. Circuito integrado para aplicaciones específicas.
ATR	Automatic Target Recognition. Reconocimiento automático de objetivo.
BRAM	Ramdom Access Memory Block. Bloque de memoria de acceso aleatorio.
DIP	Digital Images Processing. Procesamiento digital de imágenes.
DSP	Digital Signal Processor. Procesador digital de señales.
EDA	Electronic Design Automation. Diseño electrónico automático.
EEPROM	Electrical Erasable and Programable Read Only Memory. Memoria de sólo lectura borrable y grabable eléctricamente.
FIFO	First In First Out. Cola de datos donde el primero que entra es el primero que sale.
FIR	Finite Impulse Response. Respuesta finita al impulso.
FPGA	Field Programable Gate Arrays. Arreglos de compuertas programables.
GME	Graphical Model Editor. Herramienta de diseño basado en modelos del fabricante ISIS.
GPP	General Purpose Processors. Procesador de propósito general.
HD	High Definition. Alta definición.
HDL	Hardware Description Language. Lenguaje de descripción de hardware.

HW	Hardware. Se refiere a los dispositivos físicos de un sistema de cómputo.
IOB	Input/Output Buffer. Almacenador de entrada/salida.
JTAG	Joint Test Action Group.
LSB	Less Significant Bit. Bit menos significativo.
MAC	Multiply and Accumulate. Multiplicación y acumulado.
MM	Mathematical Morphology. Morfología matemática.
MSE	Mean Squared normalized Error. Error cuadrático medio normalizado.
RS-232	Recommended Standard 232. Estándar para la comunicación serie entre un equipo terminal de datos (DTE) y un equipo de comunicación de datos (DCE).
SD	Standard Definition. Definición estándar.
SE	Structural Element. Elemento estructural.
Signed 14:6	Número de punto fijo con signo representado en 14 bits de los cuales 6 bits representan la parte decimal.
Signed 16:0	Número entero con signo representado en 16 bits.
Signed 8:0	Número entero con signo representado en 8 bits.
Signed 8:6	Número de punto fijo con signo representado en 8 bits, de los cuales 6 bits representan la parte decimal.
SoC	System on Chip. Sistema en un mismo encapsulado.
USB	Universal Serial Bus. Bus serie universal.
Usigned 8:0	Número entero sin signo representado en 8 bits.
XSG	Xilinx System Generator.