RIELAC, Vol.XXXII 3/2011 p.56-69 Septiembre - Diciembre ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Diseño de bloques de convolución para procesado de imágenes con FPGA

Luis Manuel Garcés Socarrás
Alejandro José Cabrera Sarmiento
Santiago Sánchez Solano
Piedad Brox Jiménez

RESUMEN / *ABSTRACT*

Este trabajo analiza distintas opciones de realización de bloques para procesado lineal de imágenes implementados sobre FPGA, así como los efectos de la elección de diferentes parámetros de diseño. Los bloques han sido desarrollados empleando un flujo de diseño basado en modelos que se apoya en el entorno MATLAB/Simulink y la herramienta System Generator de Xilinx. Su implementación física se ha llevado a cabo sobre una placa de desarrollo Spartan-3A DSP 1800 de Xilinx.

Palabras claves: convolución, FPGA, procesamiento de imágenes, System Generator

Convolution blocks' design for images' processing with FPGAs.

This work analyzes different blocks design for lineal image processing with FPGAs, and the consequences of the selection of some designs parameters. The blocks have been development using the model based design flow with MATLAB/Simulink environment and the Xilinx's tool System Generator. The different blocks have been implemented using a Xilinx's Spartan-3A DSP 1800 development board.

Key words: convolution, FPGAs, images processing, System Generator

INTRODUCCIÓN

Los algoritmos de procesamiento digital de imágenes 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 estos algoritmos en sistemas empotrados con recursos de cálculo limitados requiere la aplicación de métodos que permitan acelerar su ejecución. El avance de las tecnologías de fabricación de circuitos integrados ha permitido el desarrollo de dispositivos lógicos programables (FPGA) de elevadas prestaciones que proporcionan una solución alternativa a la implementación eficiente de este tipo de algoritmos, permitiendo asignar tareas de cálculo intensivas al hardware y explotar el paralelismo de los algoritmos [1].

Muchos algoritmos de procesado de imágenes utilizan técnicas de convolución para realizar diferentes tareas de acondicionamiento y mejora de la información pictórica con el objetivo de facilitar, tanto la interpretación humana, como el procesado, almacenamiento y transmisión de imágenes mediante sistemas de percepción autónomos [2].

La implementación de este tipo de algoritmos sobre hardware reconfigurable ha encontrado aplicación práctica en diferentes áreas, siendo una necesidad común la rapidez en la ejecución de los mismos [2-7].

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

La integración de las herramientas de diseño electrónico automatizado (EDA) con entornos de desarrollo basados en modelos ha impulsado el auge de nuevas técnicas de diseño para dispositivos reconfigurables, facilitando la implementación de sistemas de procesado modulares y autónomos [8]. System Generator (XSG) es una herramienta provista por Xilinx que facilita el diseño e implementación sobre FPGA de sistemas de procesado digital a partir de su descripción mediante modelos MATLAB/Simulik [4].

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]. La biblioteca de módulos disponible en System Generator proporciona un bloque de convolución genérico para procesado de imágenes que implementa las operaciones de multiplicación y acumulación de los píxeles de la imagen con un núcleo o kernel de convolución de 5×5 [9]. La alternativa propuesta por A. Toledo plantea el desarrollo de sistemas de procesado de imágenes utilizando bloques hardware y bloques software [10]. Los parámetros de configuración de estos bloques permiten la selección de la precisión de los bloques de cálculo, así como la latencia y el período de muestreo.

En este artículo se describe el empleo de técnicas de diseño basadas en modelos, en combinación con la herramienta XSG, para evaluar distintas opciones de implementación de bloques configurables de convolución de imágenes y se analizan los efectos (en cuanto a consumo de recursos del dispositivo programable y velocidad de operación) relacionados con la configuración de los distintos parámetros de diseño.

CONVOLUCIÓN DE IMÁGENES

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 kernel de convolución [2]. La definición matemática del algoritmo de convolución en dos dimensiones permite explotar las posibilidades de paralelismo de ejecución y el diseño de filtros específicos para kernels predefinidos.

FUNDAMENTOS MATEMÁTICOS

Dada la imagen A de $m\times n$ píxeles representada en la ecuación 1) y la matriz de convolución C de $h\times h$ elementos y radio $r=(h-1)/2$ (donde h es 3, 5, 7 ó 9) dada en 2), se define la convolución, A_c, entre la imagen A y el kernel C mediante la operación matemática representada por la ecuación 3), 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^o (C'). Para la ecuación 2), $\alpha=r-i+1$ y $\beta=r-j+1$. Además, se excluyen los bordes de la imagen ($r<i<m-r$ y $r<j<n-r$) donde no existe la información necesaria para realizar el cálculo. Como resultado de esta operación se obtiene una imagen procesada en todos los puntos menos en los bordes de la misma [2].

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \qquad 1)$$

$$C=\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}. \qquad 2)$$

$$A_c(i-r,j-r)=\sum_{k=i-r}^{i+r}\sum_{l=j-r}^{j+r}A(k,l)C'(k+\alpha,l+\beta) \qquad 3)$$

donde $A_c(i-r, j-r)$ es la imagen resultante; $A(k,l)$ es la imagen original; y $C'(k+\alpha, l+\beta)$ es el kernel de convolución rotado 180^o. Los kernels de convolución utilizados en muchos algoritmos de procesado de imágenes cumplen con la característica de ser simétricos con respecto a la diagonal principal $(c_{ij} = c_{ji}, i \neq j)$. La utilización de estos kernels simétricos permite la reducción del número de multiplicadores, de h^2 a $(h^2 - h(h-1)/2)$, en la operación de convolución [11].

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 los coeficientes del kernel dada por:

$$S = \sum_{i=1}^{h} \sum_{j=1}^{h} C_{ij} \qquad 4)$$

donde: S es la suma de los elementos del kernel y C_{ij} es el kernel de convolución.

El proceso de normalización no puede realizarse de esta manera cuando el resultado de la ecuación 4) es el valor 0. Como alternativa en este caso se puede optar por no normalizar los coeficientes del kernel de convolución o por normalizarlos mediante el valor máximo entre la suma de todos los valores positivos y la suma de todos los valores negativos. Los efectos en la normalización del kernel están ligados a la cantidad de información que se muestra en la imagen resultante, obteniéndose imágenes saturadas (Figura 1a), donde existe una gran cantidad de información por encima del nivel máximo (255 para imágenes en escalas de grises en 8 bits) que es interpretada como blanco, e imágenes normalizadas (Figura 1b), donde se mantienen los niveles originales.

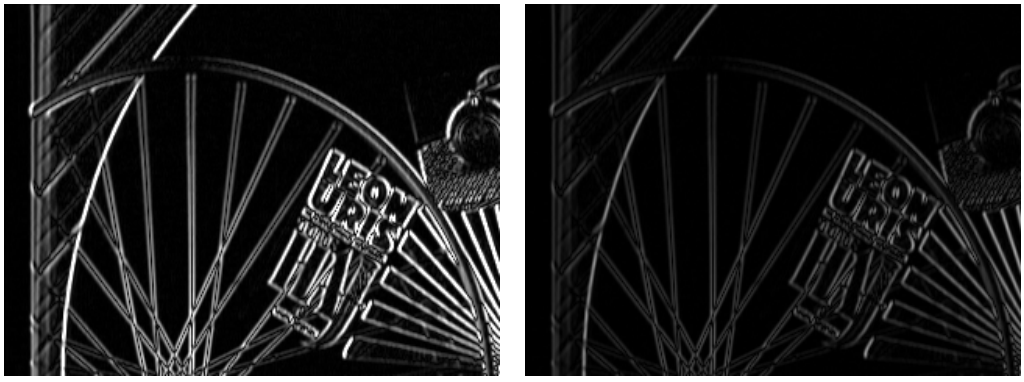

Figura 1: Filtro Prewitt X a) Salida saturada b) Salida normalizada

IMPLEMENTACIÓN SOBRE HARDWARE RECONFIGURABLE

La implementación de algoritmos de convolución sobre hardware reconfigurable permite aumentar la velocidad de ejecución con respecto a las soluciones software. La estructura de arreglos de compuertas y registros en paralelo de los FPGA hacen a los mismos una opción viable para explotar el paralelismo de datos de las imágenes o tramas de vídeos. Éstos pueden ser utilizados para realizar operaciones completas o para preprocesar los datos antes de enviarlos a un procesador digital de señales (DSP) estándar o a un microprocesador [12]. No obstante, su uso suele estar condicionado por la disponibilidad de recursos en el dispositivo reconfigurable utilizado.

En este sentido, el diseño de algoritmos de convolución basados en la definición matemática de la función permite explotar el paralelismo del algoritmo, aunque puede ocupar un porcentaje elevado de los bloques de multiplicación dedicados del FPGA. La operación de convolución definida en la ecuación 3) muestra una total independencia de los píxeles en la ventana en el algoritmo. La **¡Error! No se encuentra el origen de la referencia.** muestra varias arquitecturas para el algoritmo descrito. La La **¡Error! No se encuentra el origen de la referencia.**a expone la versión genérica del algoritmo de convolución. Luego del almacenamiento de los píxeles en la ventana, los mismos son multiplicados por el respectivo valor del kernel de convolución rotado 180º, para luego sumar todos los resultados parciales. Esta arquitectura presenta un consumo elevado de los multiplicadores dedicados del FPGA, siendo esto un elemento a minimizar en cualquier diseño. Este porcentaje se reduce en el caso de kernels de convolución simétricos.

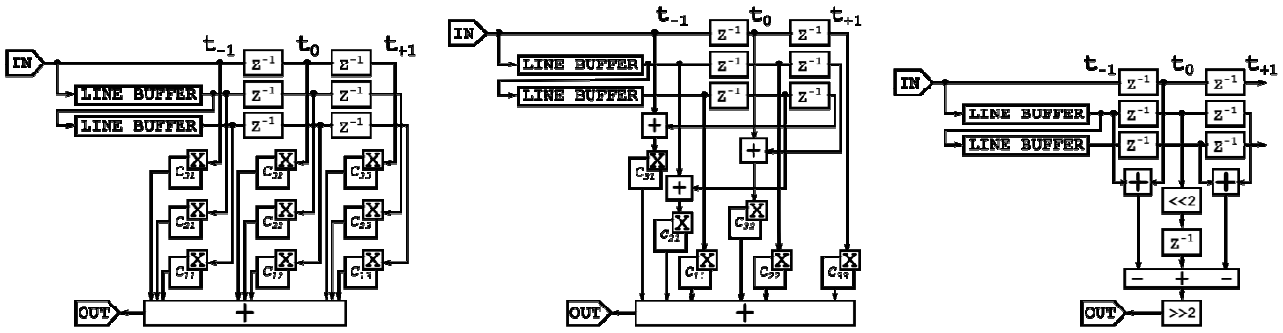

Figura 2: Implementación de filtros de convolución 3x3: a) Genérico. b) Simétrico. c) Específico

En la ¡Error! No se encuentra el origen de la referencia.b b son sustituidos $h(h-1)/2$ multiplicadores del diseño por bloques de suma, lo que permite reducir la cantidad de bloques dedicados aumentando los bloques de lógica. Los píxeles simétricos a la diagonal principal son sumados antes de ser multiplicados por el coeficiente del kernel.

Como alternativa, el diseño de módulos de convolución como el de la ¡Error! No se encuentra el origen de la referencia.c permite disponer de bloques específicos, optimizados para una determinada aplicación. De acuerdo con los valores de un kernel dado, es posible reducir el consumo de recursos optimizando la arquitectura para esa operación. La sustitución de multiplicadores dedicados por rotaciones y sumas posibilita mejorar el diseño en cuanto a velocidad de ejecución y consumo de recursos. El inconveniente de estas unidades es la necesidad de intercambiar la misma para realizar una nueva operación en el sistema de procesado.

FLUJO DE DISEÑO CON XILINX SYSTEM GENERATOR

System Generator (XSG) es una herramienta provista por Xilinx e integrada al ambiente de desarrollo de MATLAB/Simulink, que permite el desarrollo 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 [4, 5, 13]. 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.

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 3, 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

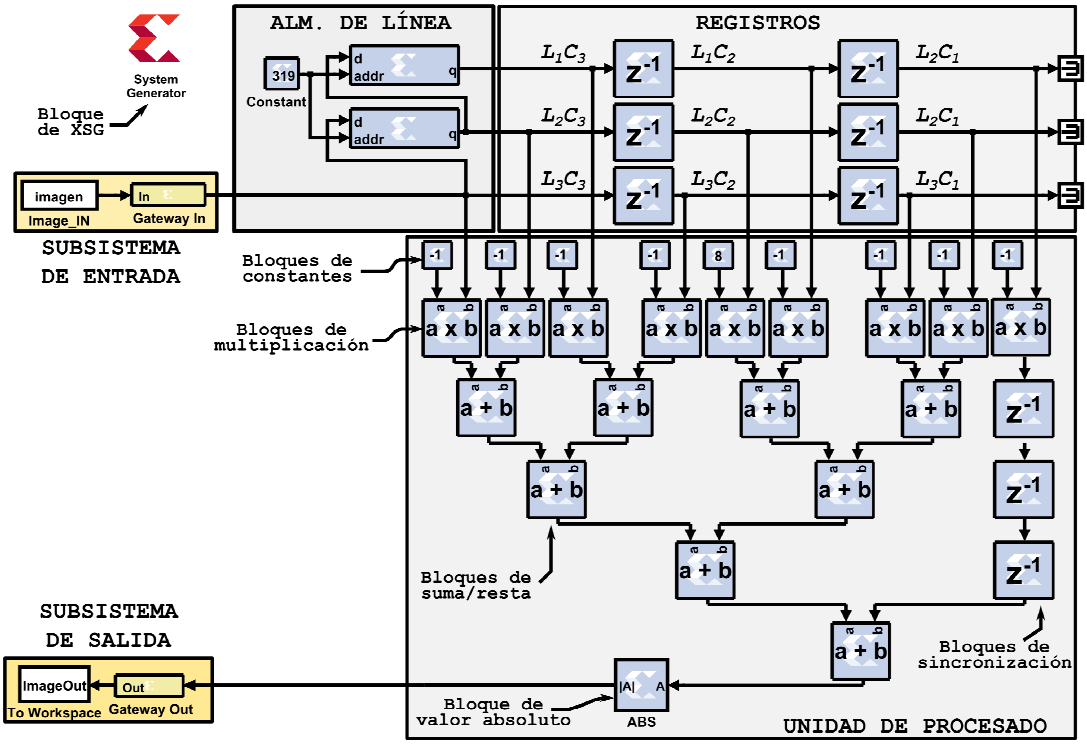

Figura 3: Sistema de procesado de imágenes con XSG.

biblioteca de System Generator, siendo además la sección del diseño a implementar sobre hardware reconfigurable.

El sistema, que se detalla en los próximos párrafos, es tomado como patrón para el diseño, simulación, co-simulación y comprobación de los bloques de procesado lineal, cuyos resultados son descritos en secciones 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 de ser necesario) y la creación de un vector de entrada con los datos de la imagen serie (para 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.

En todo sistema de diseño para XSG debe incluirse el bloque de procesado de System Generator. 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.

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 (dependiendo de la cantidad de entradas del diseño) bloques *Gateway In* (puerta de entrada), delante de los cuales se encuentran estructuras o bloques de Simulink para la entrada de información al sistema. De forma similar, el subsistema de salida comienza con uno o varios (dependiendo de la cantidad de salidas del diseño) bloques *Gateway Out* (puerta de salida) seguidos de bloques de Simulink para la recepción y el procesado de los resultados por MATLAB.

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 de salida. Este subsistema es el más importante del diseño ya que, a partir de él, 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 por cada canal 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 3) al sistema de procesado (subsistema de procesado en la Figura 3) y visualización (subsistema de salida en la Figura 3). 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 [2, 7, 14, 15, 16]. Los valores paralelos de los píxeles de la imagen en la ventana son las señales $L_a C_b$ (con $1 \le a \le h$ y $1 \le b \le h$, donde h es el tamaño de la ventana) siendo L_h y C_h la fila y columna más recientes respectivamente.

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 de salida. Los registros se componen de bloques básicos de XSG, demorando la entrada un período de reloj.

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 3, es la encargada de realizar el algoritmo matemático sobre la imagen. Ésta se compone 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).

Los bloques de constantes (**¡Error! No se encuentra el origen de la referencia.**) almacenan los valores del kernel de convolución. El mismo es introducido manualmente o seleccionado mediante la lista desplegable de la ventana de configuración del bloque de procesado. La configuración de estos bloques permite la selección de la representación del valor, así como otros parámetros. 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.

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 otros). Los bloques de multiplicación (**¡Error! No se encuentra el origen de la referencia.**) pueden utilizar bloques de multiplicación dedicados del dispositivo programable o recursos lógicos generales del mismo. La utilización de bloques de multiplicación dedicados permite incrementar la frecuencia de la operación, mientras que el uso de bloques lógicos posibilita el ahorro de los recursos específicos del dispositivo programable.

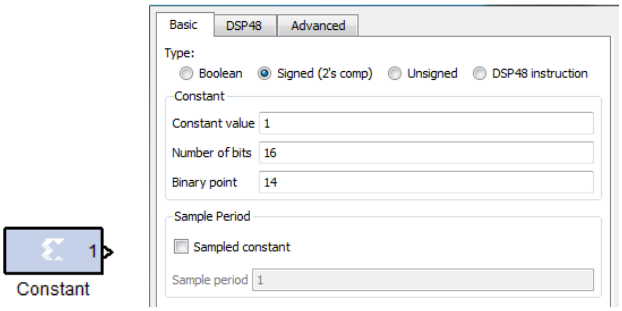

Figura 4: Bloque de constante de XSG. a) Diagrama. b) Propiedades.

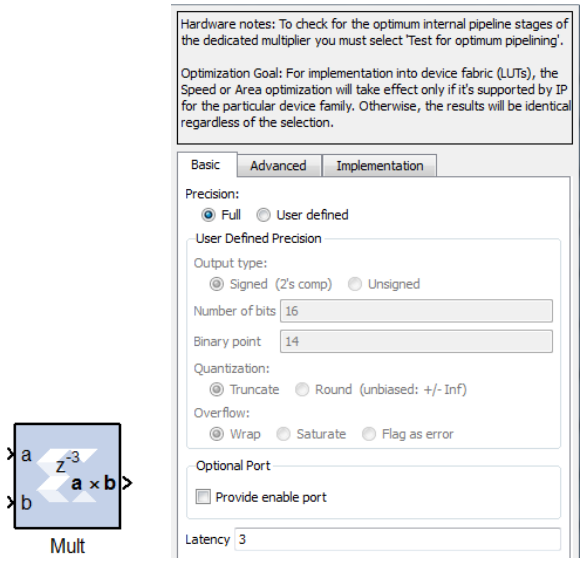

Figura 5: Bloque de multiplicación de XSG. a) Diagrama. b) Propiedades.

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.

Dado 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. Esto 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 de salida al espacio de trabajo de MATLAB, se procede a la comprobación del diseño realizado. 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.

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 6) 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 FPGA de la placa conectada, permitiendo que el flujo de entrada de datos fluya desde el espacio de trabajo de MATLAB a la misma. La información es procesada dentro del hardware del FPGA y devuelta al entorno de trabajo de MATLAB.

Figura 6: Co-simulación hardware del sistema de procesado.

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 en esta última influye el posicionado y ruteo de los componentes lógicos y específicos en el dispositivo físico.

CONVOLUCIÓN DE IMÁGENES CON XILINX 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 brinda una serie de bloques básicos, configurables en gran medida, para el desarrollo de sistemas de procesado más complejos. Además, presenta un módulo de convolución parametrizable basado en bloques de multiplicación y acumulado (*MAC*) para ventanas de 5×5 [9].

La utilización de recursos específicos hardware (como los bloques *DSP48A* disponibles en los FPGA *Spartan-3A DSP*) permite optimizar la implementación de los algoritmos de convolución. Sin embargo, debido al carácter limitado de este tipo de recursos, es importante reducir su uso al mínimo imprescindible. Una estrategia para conseguir este objetivo consiste en explotar la simetría de algunos kernels predefinidos para reducir la cantidad de multiplicadores utilizados en el diseño [2, 6, 7].

Incluso utilizando bloques específicos, las operaciones de multiplicación requieren varios ciclos de reloj. La configuración de este tiempo con un valor óptimo permite minimizar el período de reloj del sistema. En caso contrario el sistema debe esperar a que el cálculo de la operación finalice, aumentando el periodo de la señal de reloj y disminuyendo la frecuencia de ejecución. Además, esta demora hace necesaria la sincronización de los restantes elementos en la arquitectura utilizando bloques de retardo en el 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.

Considerando imágenes en escala de grises con valores cuantificados en 8 bits, los píxeles de la imagen se representan en valores enteros positivos. La normalización del kernel de convolución, sin embargo, implica la necesidad de representar los coeficientes con valores decimales, entre 0 y 1, y el uso de aritmética de punto fijo en las operaciones matemáticas del algoritmo para ajustar la salida a los niveles de la imagen original.
El número de bits utilizados para representar los valores decimales, tanto de los coeficientes del kernel como del resultado de los cálculos matemáticos intermedios del algoritmo, es un parámetro que puede ser empleado para reducir los recursos utilizados en el dispositivo [10]. 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 **¡Error! No se encuentra el origen de la referencia.** muestra el resultado de dos operaciones de convolución aplicando esta parametrización. Visualmente los cambios en las dos imágenes son imperceptibles, aunque los cálculos realizados con respecto a la imagen procesada por software demuestran la existencia de diferencias entre las mismas. Esta diferencia, obtenida al normalizar el kernel y las operaciones intermedias, es cuantificable calculando el error cuadrático medio normalizado (MSE) de la diferencia entre la imagen procesada con toda la precisión de cálculo y la correspondiente a uso de una precisión limitada. El MSE debe ser valorado por el usuario para decidir la parametrización o no de este elemento según sus necesidades.

Por último, 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. 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.

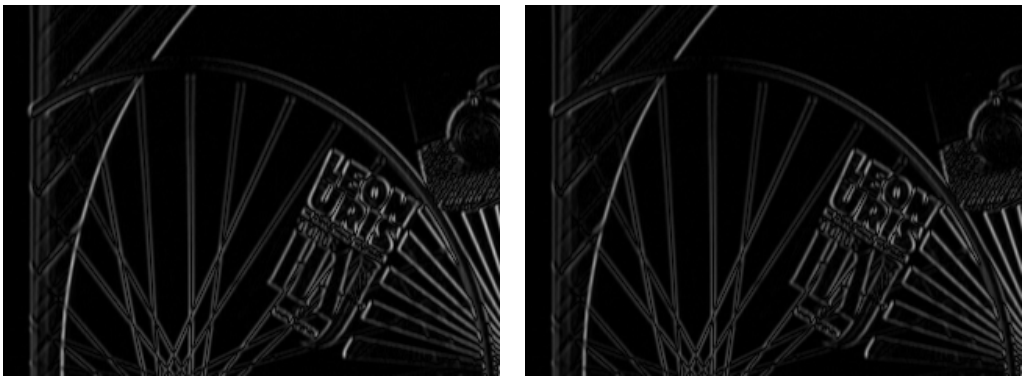

Figura 7: Filtro Prewitt X a) 3 bits decimales MSE 9,8 b) 8 bits decimales MSE 0,029

En las siguientes secciones se analizan los resultados obtenidos al considerar diferentes opciones y parámetros de configuración en los bloques de convolución diseñados con System Generator. Los resultados reportados tienen en cuenta la utilización de kernels simétricos, el ajuste de los ciclos de demora de los multiplicadores, la precisión utilizada en el cálculo, y el uso de filtros específicos.

RESULTADOS OBTENIDOS

La aplicación de las consideraciones anteriores en el diseño de bloques de convolución con XSG permite el desarrollo de una biblioteca de módulos para procesado de imágenes que incluye diferentes elementos configurables por el usuario en función de las necesidades de su aplicación.

Los resultados mostrados a continuación ilustran los efectos de las distintas opciones y parámetros de configuración en la implementación sobre una placa de desarrollo Spartan-3A DSP 1800, de distintos bloques para procesado lineal de imágenes.

KERNEL SIMÉTRICOS VS. NO SIMÉTRICOS

La utilización de kernels de convolución simétricos permite reducir la utilización de recursos del dispositivo. El número de multiplicadores pasa de 9 a 6 para un kernel 3×3 y de 25 a 15 para un kernel 5×5. Como se muestra en la Tabla 1, esto representa un ahorro del 33,33% de los recursos específicos para el kernel de 3×3. El número de Slices (menor unidad de recursos lógicos en el FPGA) también se reduce aunque no considerablemente.

Tabla 1: Utilización de recursos: convolución no simétrica vs. simétrica

Recursos	No simétrica	Simétrica
Slices	215	176
DSP48A	9	6
Frecuencia	182,216 MHz	181,389 MHz

OPTIMIZACIÓN EN VELOCIDAD VS ÁREA

El bloque de multiplicación de XSG, mostrado en la **¡Error! No se encuentra el origen de la referencia.**a permite la ejecución de la operación deseada en el sistema [9]. Este bloque permite la configuración de la latencia de la operación entre otros parámetros, como muestra la **¡Error! No se encuentra el origen de la referencia.**b, en el campo *"Latency"* [9]. La configuración del número de ciclos de espera en los multiplicadores permite optimizar el algoritmo de convolución en función del consumo de recursos o la velocidad de ejecución. Además, éste posibilita el cálculo de la latencia óptima para el sistema de procesado mediante la simulación del mismo. Métodos experimentales arrojan una latencia óptima de 4 ciclos de reloj mejorando la frecuencia de ejecución (optimización en velocidad), mientras que un valor menor de 3 disminuye la misma, además del consumo de recursos lógicos (optimización en área).

La Tabla 2 muestra que la cantidad de Slices utilizados al aplicar el criterio de optimización en área se reduce en un 34,42%, mientras que la frecuencia de trabajo se incrementa un 33,93% cuando se aplica el criterio de optimización en velocidad.

NORMALIZACIÓN DE LAS OPERACIONES

El bloque de constantes de XSG, mostrado en la **¡Error! No se encuentra el origen de la referencia.**a, así como la mayoría de los bloques de la biblioteca básica que trabajan con valores numéricos, permiten la selección de la cantidad de bits para representar los mismos. La **¡Error! No se encuentra el origen de la referencia.**a y la **¡Error! No se encuentra el origen de la referencia.**b exponen algunos de éstos parámetros para un bloque de cálculo, como para el bloque de constantes respectivamente [9]. En este último, la representación del valor numérico (campo *"Constant value"*) necesita ser configurado. Entre los parámetros de configuración están: el valor (campo *"Value"*); la cantidad de bits del mismo (campo *"Number of bits"*;, y la cantidad de bits para la parte decimal (campo *"Binary point"*). El total de bits se calcula mediante la suma de la cantidad de bits para representar la parte entera, un bit para el signo, y la cantidad de bits para la parte decimal del coeficiente. Éste último valor puede ser configurado para limitar el consumo de recursos lógicos utilizados en el diseño.

Tabla 2: Utilización de recursos: optimización en velocidad vs área

Recursos	Velocidad	Área
Slices	215	141
DSP48A	9	9
Frecuencia	182,216 MHz	136,054 MHz

NORMALIZACIÓN DE LAS OPERACIONES

El bloque de constantes de XSG, mostrado en la **¡Error! No se encuentra el origen de la referencia.**a, así como la mayoría de los bloques de la biblioteca básica que trabajan con valores numéricos, permiten la selección de la cantidad de bits para representar los mismos. La **¡Error! No se encuentra el origen de la referencia.**b y la **¡Error! No se encuentra el origen de la referencia.**b exponen algunos de éstos parámetros para un bloque de cálculo, como para el bloque de constantes respectivamente [9]. En este último, la representación del valor numérico (campo *"Constant value"*) necesita ser configurado. Entre los parámetros de configuración están: el valor (campo *"Value"*); la cantidad de bits del mismo (campo *"Number of bits"*); y la cantidad de bits para la parte decimal (campo *"Binary point"*). El total de bits se calcula mediante la suma de la cantidad de bits para representar la parte entera, un bit para el signo, y la cantidad de bits para la parte decimal del coeficiente. Éste último valor puede ser configurado para limitar el consumo de recursos lógicos utilizados en el diseño.

Las operaciones de cálculo permiten la configuración de la cantidad de bits para la representación de la salida, así como los métodos para limitar la misma (campo *"User Defined Precision"*). Al seleccionar el parámetro descrito para los bloques de constantes, se configura con el mismo valor la representación de la parte decimal del resultado (campo *"Binary point"*), truncando el valor a ésta.

La elección del número de bits adecuados para representar la parte decimal de los coeficientes del kernel normalizados incide en la cantidad de Slices utilizados en la implementación del bloque de convolución. La Tabla 3 muestra la variación en el consumo de recursos del dispositivo dependiendo de la configuración de este valor.

Tabla 3: Utilización de recursos: convolución precisa vs. normalizada

Recursos	Prec. total	Norm. 8 bits	Norm. 3 bits
Slices	215	214	154
DSP48A	9	9	9
Frecuencia	182,216 MHz	191,755 MHz	183,419 MHz
MSE	0	0.029	9.8

En el caso estudiado el error cuadrático medio provocado por la pérdida de precisión no se aprecia a simple vista (como se explicó anteriormente para la **¡Error! No se encuentra el origen de la referencia.**), aunque los valores de MSE corroboran la existencia de diferencias entre la operación normalizada y sin normalizar. 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 parámetro puede ser útil al usuario a la hora de reducir la utilización de recursos.

FILTROS GENÉRICOS VS. ESPECÍFICOS

El diseño de filtros específicos para los kernels de convolución predefinidos permite optimizar la implementación hardware del algoritmo al evitar la duplicación de recursos y sustituir los multiplicadores por operaciones de suma y desplazamiento. A modo de ejemplo, la arquitectura de un filtro específico Smooth, como muestra la Figura 8, sustituye los bloques de multiplicación dedicados por rotaciones y sumas, los cuales se implementan con los recursos lógicos del dispositivo.

De acuerdo con el kernel de convolución del filtro Smooth, es necesario realizar multiplicaciones por los valores 5 y 44, los cuales no son potencias de 2. Para la multiplicación por 5 se rota el píxel correspondiente 2 lugares a la izquierda (multiplicación por 4) para luego agregarle el valor original. La multiplicación por 44 utiliza como base la anterior descrita. Luego de multiplicar por 5 el resultado es rotado una vez a la izquierda (multiplicación por 2) y se le suma el valor original, para obtener un multiplicador por 11. Por último, éste es rotado 2 veces a la izquierda (multiplicación por 4) para obtener el resultado deseado. Luego de terminado cada cálculo intermedio, éstos son sumados utilizando bloques de suma en cascada y bloques de demoras para el sincronismo de las operaciones, minimizando el período de ejecución a costa de una latencia en el procesado.

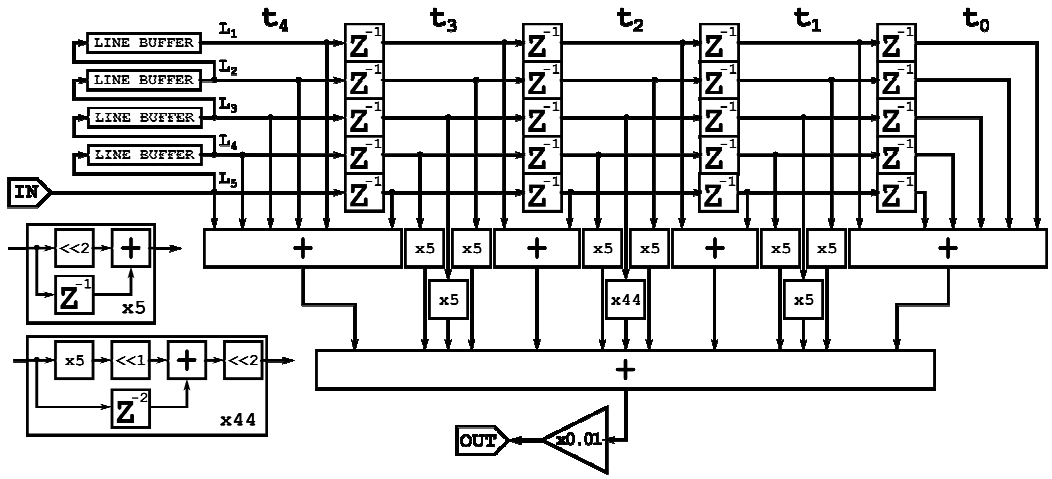

Figura 8: Arquitectura del filtro específico Smooth

La Tabla 4 recoge la comparación entre un filtro Smooth y un algoritmo genérico de 5×5. En ésta se muestra la supresión de los multiplicadores, la reducción de la cantidad de Slices en un 26,56% y el aumento de la frecuencia de trabajo en un 25,12%.

Tabla 4: Utilización de recursos: algoritmo genérico 5x5 vs. filtro específico Smooth

Recursos	Genérico	Smoth
Slices	433	318
DSP48A	15	0
Frecuencia	168,604 MHz	210,970 MHz

FILTRO DE CONVOLUCIÓN DE SYSTEM GENERATOR

Como ya se ha mencionado, XSG presenta un bloque de convolución para ventanas de 5×5. 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 [4]. El uso de bloques MAC, con 5 unidades de procesado, establece una arquitectura semiparalela 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 o el período de muestreo del bloque debe ser 5 veces mayor que un sistema completamente paralelo.

En la Tabla 5 se comparan los resultados de implementación de los bloques de convolución genéricos y específicos descritos en

Tabla 5: Utilización de recursos: algoritmo XSG 5x5, genérico 5x5 y filtro específico Gaussian

Recursos	XSG	Genérico	Gaussian
Slices	381	433	251
DSP48A	5	15	0
BRAM	5	0	0
Frecuencia	27,211 MHz	168,604 MHz	241,067 MHz

este trabajo frente al módulo de convolución genérico proporcionado por Xilinx. Puede observarse que el bloque genérico propuesto proporciona una velocidad de operación cinco veces superior al de Xilinx y evita la utilización de bloques de memoria a costa de emplear un número de Slices ligeramente superior.

La utilización de un filtro específico Gaussian permite en este caso incrementar 8,86 veces la frecuencia de operación con una reducción del 34,12% en el consumo de Slices del dispositivo.

CONCLUSIONES

Las posibilidades de configuración de los bloques de procesado de imágenes desarrollados con System Generator facilitan la optimización de los diseños y su adaptación a las necesidades del usuario.

La utilización de kernels simétricos permite un ahorro considerable de los recursos del dispositivo, disminuyendo la cantidad de multiplicadores del sistema.

La optimización en área o velocidad permite establecer un adecuado compromiso entre la velocidad de procesado o el consumo de recursos.

La selección de una determinada precisión en la representación de los coeficientes de los kernels permite disminuir la utilización de recursos proporcionando imágenes resultantes con errores cuantificables, que permiten decidir el valor de la parametrización.

Por último, el empleo de filtros específicos brinda una mejor optimización de los recursos y de la velocidad de ejecución siendo la mejor opción cuando se utilizan kernels predefinidos.

Las consideraciones anteriores están siendo tenidas en cuenta en el diseño de una biblioteca de bloques para procesado de imágenes actualmente bajo desarrollo.

RECONOCIMIENTOS

Los autores del trabajo agradecen a la Agencia Española de Cooperación Internacional para el Desarrollo (AECID) por financiar parcialmente este trabajo en el marco de los proyectos PCI FORTIN, Ref. D/024/124/09 y Ref. D/030769/10.

REFERENCIAS

1. **QUASIM, S.M.; ABBASI, S. and ALMASHARY, B.:** "An overview of advanced FPGA architectures for optimized hardware realization of computation intensive algorithms": *International Multimedia, Signal Processing and Communication Technologies*, pp. 300-303, 2009. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5164235
2. **GONZÁLEZ, R.C. and WOODS, R.E.:** *"Digital image processing" 2nd Edition. Ed.Prebtice Hall.* New Jersey, USA pp 67-70, 116-134, 205-208, 283-302, 308-313, 2002. http://portal.acm.org/citation.cfm?id=130248
3. **BENKRID K.; and BELKACEMI, S.:** *"Design and implementation of a 2D convolution core for video applications on FPGAs": Third International Workshop on Digital and Computational Video*, pp. 85-92, 2002. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1218747
4. **RODRÍGUEZ CRUZ, C.; RIVERO FLORES, R.; CASTILLO ATOCHE, A. and VÁZQUEZ CASTILLO J.:** *"Procesamiento de Imágenes con Xilinx System generator" Primer Congreso Internacional de Sistemas Computacionales y electrónicos*, pp. 1-7, 2006.
5. **SÁNCHEZ G., A.M.; ALVAREZ G., R. and SÁNCHEZ G., S.:** *"Architecture for filtering images using Xilinx system generator," in International Journal of Mathematics and Computer in Simulation*, vol. 1, no. 2, pp. 101–107, 2007.
6. **TOLEDO MORENO, A.; VICENTE-CHICOTE, C.; SUARDÍZ MURO, J. and CUENCA ASENSI, S.:** *"Xilinx System Generator Based HW Components for Rapid Prototyping of Computer Vision SW/HW Systems," Pattern Recognition and Image Analysis*, pp. 667–674, 2005. http://www.springerlink.com/index/grjl91g4q100cbu7.pdf
7. **WNUK, M.:** *"Remarks on Hardware Implementation of Image Processing Algorithms," International Journal of Applied Mathematics and Computer Science*, vol. 18, no. 1, pp. 105–110, 2008. http://versita.metapress.com/openurl.asp?genre=article&id=doi:10.2478/v10006-008-0010-2

8. **SANGIOVANNI-VINCENTELLI, A.:** *"The tides of EDA," Design & Test of Computers, IEEE,* vol. 20, no. 6, pp. 59–75, 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1246165
9. **XILINX:** *"Xilinx System Generator for DSP Reference Guide"*, pp. 379–380, 2010.
10. **TOLEDO MORENO, A.; SUARDÍZ MURO, J. and CUENCA ASENSI, S.:** *"Entorno de codiseño para sistemas heterogéneos de procesamiento de imagen."* http://www.aurova.ua.es:8080/ja2005/comu/4511-Cuenca_ja05word.pdf
11. **TURNEY, R.:** *"Two-Dimensional Linear Filtering," Xilinx Application Notes*, vol. 933, pp. 1–8, 2007.
12. **JOHNSTON, C.T.; BAILEY, D.G.; LYONS, P.:** *"A Visual Environment for Real-Time Image Processing in Hardware," EURASIP Journal on Embedded Systems 2006* pp. 1–8, 2006.
13. **SAIDANI, T.; DIA, D.; ELHAMZI, W.; ATRI, M.; TOURKI, R.:** *"Hardware Co-simulation For Video Processing Using Xilinx System Generator," Proceedings of the World Congress on Engineering*, vol. 1 pp. 3–7, 2009.
14. **ADARIO, A.; ROEHE, E.; BAMPI, S.:** *"Dynamucally reconfigurable architecture for image processor applications", Proceedings 1999 Design Automotion Conference* pp. 632-628, 1999.
15. **BANDARI, S.; PUJARI, S.; SUBBARAMAN, S.; MAHAJAN, R.:** *"Real Time Video Processing on FPGA Using on the Fly Parcial Reconfiguration", 2009 International Conference on Signal Processing System*, pp. 245-247, 2009.
16. **OSIO, J.; RAPALLINI, J.; QIJANO, A.; OCAMPO, J.:** *"Implementación de un algoritmo para procesamiento de imágenes en una FPGA", Congreso de Microelectrónica Aplicada 2010*, pp. 38-42, 2010.

AUTORES

Luis Manuel Garcés Socarrás, Ingeniero en Automática, Máster en Sistemas Digitales, Profesor Instructor, Departamento de Automática y Computación del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana. Actualmente desarrolla su tema de investigación pre-doctoral sobre aceleración de algoritmos para el procesado de visión mediante hardware reconfigurable.

e-mail: lmgarcess@electrica.cujae.edu.cu

Alejandro José Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas, Profesor Titular, Departamento de Automática y Computación del Instituto Superior Politécnico "José Antonio Echeverría", calle 114 Nº 11901, CUJAE, Marianao, La Habana.

e-mail: alex@electrica.cujae.edu.cu.

Santiago Sánchez Solano, Doctor en Física por la Universidad de Sevilla en 1990, Investigador Principal del CSIC adscrito al Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica (IMSE-CNM-CSIC), Sevilla, España,

e-mail: santiago@imse-cnm.csic.es

Piedad Brox Jiménez, Doctora en Física por la Universidad de Sevilla en 2009, Investigadora postdoctoral del Programa Nacional Juan de la Cierva adscrita al Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica (IMSE-CNM-CSIC), Sevilla, España

e-mail: brox@imse-cnm.csic.es