Self-Modifiable Image Processing Library for Model-Based Design on FPGAs

L. Garcés-Socarrás, A. Cabrera, S. Sánchez-Solano, P. Brox, E. Ieno, and T. Pimenta

***Abstract*—This paper describes highly configurable hardware modules, included in *XIL XSGImgLib* library, capable of speed up the hardware implementation of video and image processing systems using the model-based design flow provided by Xilinx System Generator. As part of this work, generic architectures were developed to exploit specific characteristics of some processing blocks, which can be self-modified using a novel software procedure developed for MATLAB®. This procedure, along with the generic architecture and the configuration options, allows the abstraction about the specific details of the hardware implementation, as well as the adjustment of the resources consumption of the high-speed image and video processing application for embedded systems with weight, volume and power consumption constrains like smart cameras, video surveillance and autonomous vehicles. The use of this video and image processing library is illustrated by the development of a segmentation application on a *Spartan-6 LX45* FPGA although any Xilinx's FPGA is supported.**

***Index Terms*—Video and Image Processing, XIL XSGImgLib, FPGA, Xilinx System Generator, Model-based design flow, MATLAB®/Simulink®.**

I. Introducción

Los algoritmos de visión computacional realizan transformaciones de los datos de una imagen o vídeo para generar una nueva representación o decisión. En el ser humano estas tareas son realizadas por el cerebro, el cual es el encargado de analizar las señales entregadas por el sistema visual, dividiéndolas en muchos canales con varios tipos de información. En un sistema de visión computacional (CVS: *Computational Vision System*) estas operaciones son realizadas por una unidad de procesado, la cual recibe la secuencia de imágenes como una serie de datos numéricos para realizar su análisis [1]–[5].

L. M. Garcés-Socarrás, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, Cuba, lmgarcess@automatica.cujae.edu.cu.

A. J. Cabrera, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, Cuba, alex@automatica.cujae.edu.cu.

S. Sánchez-Solano, Instituto de Microelectrónica de Sevilla, IMSE-CNM (CSIC/Universidad de Sevilla), Sevilla, España, santiago@imse-cnm.csic.es.

P. Brox, Instituto de Microelectrónica de Sevilla, IMSE-CNM (CSIC/Universidad de Sevilla), Sevilla, España, box@imse-cnm.csic.es.

E. Ieno Junior, Centro Federal de Educação Tecnológica de Minas Gerais (CEFET MG), Varginha, Minas Gerais, Brasil, egidio_ieno@yahoo.com.br.

T. C. Pimenta, Universidade Federal de Itajubá (UNIFEI), Itajubá, Minas Gerais, Brasil, tales@unifei.edu.br.

La unidad de procesado es la parte más importante de un CVS. Pueden emplearse tres plataformas principales para su implementación: secuenciales, que ejecutan software codificado en instrucciones; paralelas, donde el algoritmo se lleva a cabo a través de hardware; e híbridas, que vinculan las anteriores. Estas plataformas de implementación son seleccionadas para asegurar el cumplimiento de los términos de desarrollo: el tiempo empleado para obtener la aplicación, los requisitos funcionales y las restricciones temporales [2], [6]–[14].

Cuando las aplicaciones a desarrollar requieran de alta velocidad de procesado, pero presenten restricciones de costo, consumo de potencia, autonomía, portabilidad y tamaño (como puede ser el caso de cámaras autónomas para tareas de vigilancia o control de producción), la implementación sobre una computadora personal (PC: *Personal Computer*) es inviable, precisando su desarrollo sobre dispositivos empotrados. Los dispositivos empotrados secuenciales, como por ejemplo los microcontroladores y los procesadores digitales de señales (DSP: *Digital Signal Processor*), presentan generalmente restricciones de velocidad, por lo que es preciso el empleo de plataformas paralelas. Descartando a los circuitos integrados de aplicaciones específicas (ASIC: *Application-Specific Integrated Circuit*) por sus altos costos de diseño e implementación, son los arreglos de compuertas lógicas programables (FPGA: *Field-Programable Gates Array*) los dispositivos idóneos para el desarrollo de aplicaciones con las restricciones señaladas [5], [8], [11]–[13], [15].

La implementación de algoritmos directamente sobre hardware es una tarea ardua y compleja, donde es preciso un dominio sobre la plataforma de implementación y las herramientas de desarrollo. Una forma de mitigar este inconveniente es mediante el uso de bibliotecas de procesado de imágenes y vídeos configurables para dispositivos hardware, así como el aumento del nivel de abstracción de las herramientas de diseño electrónico, de forma tal que faciliten la implementación de sistemas de procesado complejos de una forma rápida y poco costosa, liberando al diseñador de los detalles específicos del hardware de los diferentes componentes [1], [5], [7].

El diseño de sistemas de procesado para hardware se basa en tres flujos principales: el diseño convencional basado en lenguajes de descripción de hardware (HDL: *Hardware Description Language*), la síntesis de alto nivel (HLS: *High-Level Synthesis*) y el flujo de diseño basado en modelos.

La creación de sistemas de visión computacional basados en HDL necesita de un mayor dominio del desarrollo de sistemas empotrados sobre FPGA por parte del diseñador. Además, las

bibliotecas para este flujo de diseño pueden estar asociadas a un dispositivo hardware específico o carecen de suficientes opciones de configuración para reducir el consumo de recursos de hardware, o requieren del pago adicional de una licencia para la utilización de cada uno de los bloques [16]–[19].

El máximo exponente hasta el momento del incremento del nivel de abstracción de los desarrolladores es la síntesis de alto nivel, permitiendo ambientes más amigables a usuarios menos conocedores del diseño hardware, pero con un menor control sobre el consumo de los recursos de la plataforma [6], [20]–[23].

Un nivel de abstracción intermedio se obtiene con el diseño basado en modelos, el cual proporciona una estrategia de desarrollo de sistemas empotrados sobre FPGA que permite la fácil reutilización de componentes empleando ambientes de desarrollo gráficos [24].

Las bibliotecas de procesado de imágenes disponibles para el flujo de diseño basado en modelos presentan como principales inconvenientes la poca variedad de componentes; la existencia de bloques para operaciones específicas poco versátiles; o bloques genéricos poco adaptables a diferentes escenarios, así como las insuficientes opciones de configuración disponibles que permitan adecuar la aplicación al nivel de rendimiento y consumo de recursos de la plataforma hardware requeridos [6], [7], [14], [25].

Este trabajo describe el uso del flujo de diseño basado en modelos de Xilinx System Generator (XSG) para la creación de una biblioteca de bloques para tareas de procesado de imágenes y vídeos (*XIL XSGImgLib*) con arquitecturas configurables que permiten ajustar el consumo de recursos del dispositivo hardware a las necesidades de la aplicación. En la sección II se explica el procedimiento de configuración automática de las arquitecturas de los bloques de procesado. La sección III describe la biblioteca desarrollada, así como las principales modificaciones a las arquitecturas descritas en la bibliografía consultada, mientras que en la sección IV se ilustra el empleo de varios bloques de la biblioteca para la creación de un sistema de procesado de imágenes y vídeos. Por último, se presentan las conclusiones de este trabajo.

II. Procedimiento de Configuración Automática de Bloques de Procesado para *XIL XSGImgLib*

Las principales ventajas del flujo de diseño basado en modelos radican en el rápido desarrollo de prototipos, la posibilidad de modificar los parámetros y, en algunos casos, la arquitectura hardware de los bloques de procesado. Para esto último se requiere de un procedimiento que permita ajustar la estructura hardware de los diferentes bloques para adecuar la arquitectura a los requerimientos específicos de cada aplicación. Estas opciones aumentan la versatilidad de los bloques de procesado de la biblioteca *XIL XSGImgLib* y permiten la creación de sistemas de visión computacional personalizados [15].

System Generator es una herramienta de diseño de Xilinx que permite emplear el flujo de diseño basado en modelos de MATLAB®/Simulink® en la creación de aplicaciones sobre FPGA. XSG se añade como un *toolbox* de Simulink® que contiene modelos de bloques básicos de componentes hardware implementables sobre las FPGA de Xilinx. A partir de estos bloques básicos se pueden desarrollar diseños hardware completos; o también desarrollar bloques más complejos y agruparlos en una biblioteca, como se ha realizado con *XIL XSGImgLib*. El uso de esta herramienta no precisa de una experiencia previa en las metodologías de diseño hardware con los FPGA de Xilinx, modelando el diseño desde un ambiente de desarrollo amigable con módulos sintetizables disponibles en la biblioteca específica de Xilinx. Este proceso de diseño permite comprobar la funcionalidad del modelo hardware diseñado, simulando su comportamiento en el ambiente de MATLAB®/Simulink® (con todas sus ventajas inherentes); o realizando la cosimulación hardware del mismo en un FPGA, proceso que consiste en configurar el FPGA con el hardware diseñado y, desde MATLAB® /Simulink®, enviar las señales de entrada al FPGA y recibir las salidas, lo cual facilita la comprobación (y depuración) de la operación sobre el hardware real. Además, posibilita comparar los resultados de ambos procesos con la solución algorítmica desarrollada en software a total precisión sobre MATLAB®/Simulink® de la misma funcionalidad del modelo, con el objetivo de detectar y corregir errores en el modelado hardware de la aplicación. Una vez terminado el modelado de la aplicación, la herramienta XSG realiza las tareas de síntesis, mapeo, posicionado y ruteo invocando automáticamente a las herramientas de desarrollo de FPGA de Xilinx desde MATLAB®/Simulink®, obteniendo el archivo de configuración para el FPGA el cual es empleado en la implementación hardware final del diseño [26], [27].

La configuración de los bloques de procesado en el flujo de diseño basado en modelos de XSG se realiza a partir de la ventana de configuración del bloque principal, también conocida como máscara, desde la cual se parametrizan los módulos que lo componen y se realizan los cambios en la arquitectura interna del módulo en caso de ser necesarios. Para esta tarea se requiere de una arquitectura hardware genérica que permita modificar su estructura interna y un procedimiento software desarrollado para MATLAB® que efectúe estas tareas [15].

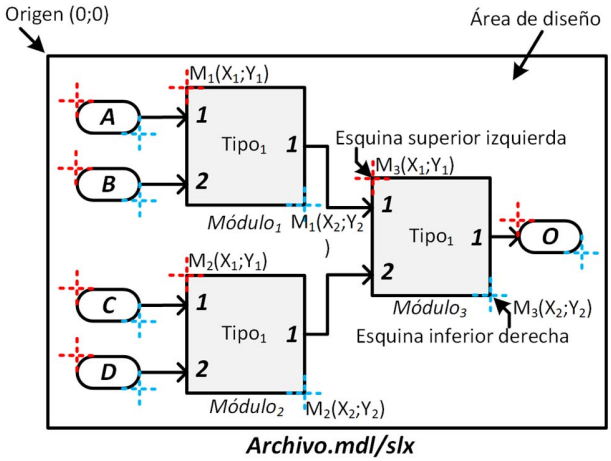


Fig. 1. Arquitectura genérica de un bloque de procesado empleando el flujo de diseño basado en modelos.

La arquitectura hardware genérica de un bloque de procesado (Fig. 1) incluye varios módulos básicos dentro del área de diseño. Cada uno de estos módulos es especificado mediante el nombre (*Módulo$_x$*), el tipo (*Tipo$_x$*), las coordenadas superior izquierda (*X1,Y1*) e inferior derecha (*X2,Y2*) y las interconexiones con los restantes módulos. El bloque se almacena en un archivo de MATLAB®/Simulink® (*Archivo.mdl/slx*) sobre el cual se realiza el proceso de modificación de la arquitectura [15].

La arquitectura genérica del bloque de procesado es analizada (Fig. 2) por un código MATLAB® (*Script de análisis.m*) que permite almacenar en el espacio de trabajo un archivo (*Bloque.mat*) con la posición y orientación de los módulos que componen el diseño. Este archivo es empleado posteriormente en el proceso de adaptación de la arquitectura a las necesidades de la aplicación [15].

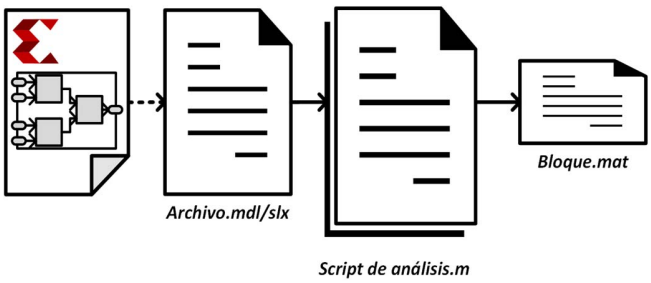


Fig. 2. Análisis de la arquitectura de un bloque de procesado.

Mediante la ventana de configuración del bloque de procesado se accede a los parámetros habilitados para adaptar el bloque genérico (Fig. 3). Los cambios realizados en la ventana de configuración ejecutan un código MATLAB® (*Script de configuración.m*) desarrollado por los autores para obtener una nueva arquitectura. En este proceso se analizan los cambios necesarios según los parámetros seleccionados en la ventana de configuración y la arquitectura genérica disponible (*Archivo.mdl/slx*) para eliminar, añadir y sustituir los módulos en la arquitectura según se precise. En estas dos últimas operaciones se emplea el archivo obtenido previamente del análisis de la arquitectura genérica (*Bloque.mat*). Los cambios realizados en la arquitectura genérica generan un nuevo archivo de modelo (*Archivo MOD.mdl/slx*) con una arquitectura hardware diferente a la original [15].

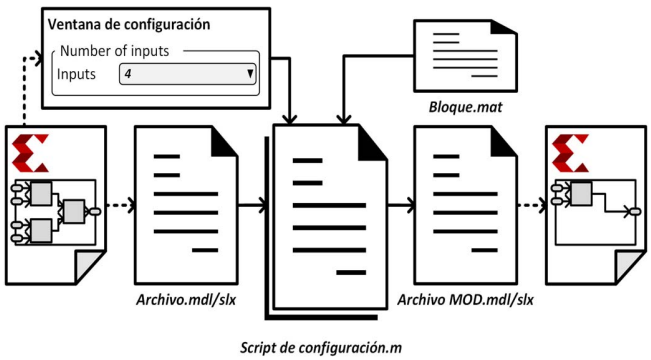


Fig. 3. Modificación de la arquitectura de un bloque de procesado.

III. Biblioteca de procesado de imágenes y vídeos *XIL XSGImgLib*

La biblioteca de procesado *XIL XSGImgLib*, disponible en el sitio web www.researchgate.net/project/XIL-XSGImgLib-Image-and-Video-processing-library-for-System-Genrator, se concibe para la rápida creación de prototipos de sistemas de procesado de imágenes y vídeos desde un ambiente de desarrollo amigable e idóneo para la comprobación de los algoritmos. Esto se debe a la facilidad de utilización y el nivel de abstracción de hardware que posee el flujo de diseño basado en modelos en unión con las herramientas MATLAB®/Simulink® y XSG. La biblioteca cuenta con un total de 54 bloques con 38 operaciones de procesado, incluidos en una serie de paletas organizadas según el tipo de operadores que contienen (Fig. 4). La implementación de cada bloque utiliza el nivel de paralelismo adecuado para ofrecer un píxel a la salida del bloque por cada píxel en su entrada después de una latencia inicial, especificada en [28]. La efectividad de cada uno de los bloques de la biblioteca se comprueba comparando el resultado de ejecución obtenido de la implementación hardware con el resultado de la implementación software equivalente a total precisión desarrollada en MATLAB®/Simulink®.

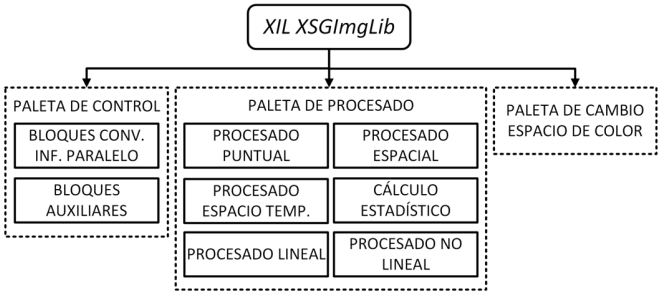


Fig. 4. Estructura de las paletas en la biblioteca *XIL XSGImgLib*.

El nivel principal se divide en tres paletas. La paleta de control contiene los bloques que permiten que la información que llega al sistema de procesado en distintos instantes de tiempo esté disponible en un momento dado, así como módulos auxiliares para el diseño, configuración e implementación de bloques de procesado más complejos. La paleta de procesado incluye la mayor parte de los bloques de la biblioteca. La última paleta abarca bloques para el cambio de los espacios de color, haciendo posible el trabajo con diferentes tipos de imágenes. Todos los componentes desarrollados para *XIL XSGImgLib* son añadidos a un archivo de biblioteca de Simulink® empleando el procedimiento definido por MATLAB®.

La biblioteca presenta como novedad la creación de 12 arquitecturas genéricas configurables empleadas en un total de 28 bloques, destacándose la convolución genérica, los filtros específicos, los operadores morfológicos y el trabajo con histogramas. También, forman parte de la biblioteca los archivos que definen las arquitecturas genéricas para cada bloque y los códigos MATLAB® desarrollados para el ajuste de la estructura interna. Además, los bloques contienen una serie de parámetros que permiten configurar los módulos de la biblioteca, entre los que se destacan el empleo de las señales de sincronismo y las señales de control; el tipo y la cantidad de bits

de los datos de entrada y salida; y las opciones de cuantificación y desbordamiento a emplear en las operaciones matemáticas y lógicas.

La paleta de procesado es la paleta principal de la biblioteca *XIL XSGImgLib*, compuesta por un total de 35 bloques configurables capaces de realizar 27 operaciones diferentes. Esta paleta contiene la mayoría de los bloques de procesado agrupados en seis subpaletas según el tipo de granularidad de las operaciones (procesado puntual, espacial y espacio temporal) o el tipo de operaciones que realizan (procesado estadístico, lineal y no lineal). Entre los principales bloques de procesado se encuentran el bloque de convolución genérico, los filtros específicos lineales, el bloque genérico de ordenamiento y los operadores morfológicos, en los cuales se realizan modificaciones a algunas de las arquitecturas propuestas en la literatura, y se emplean estructuras que pueden ser ajustadas a las necesidades de la aplicación.

Con el objetivo de ilustrar el procedimiento desarrollado para la configuración automática de diferentes bloques de la biblioteca, se exponen dos de ellos a continuación.

A. Bloque de Convolución Genérico

La convolución es la operación básica del procesado espacial lineal de imágenes, donde se transforman los $p \times q$ píxeles en una ventana de la imagen *f(x,y)* con el núcleo de convolución *w(i,j)* de la ecuación (1) que especifica las modificaciones a realizar en la imagen. Esta se define (*) como el desplazamiento del núcleo de convolución (*w*) sobre la imagen (*f*) evaluando la expresión (2) [4].

$$w(i,j) = \begin{bmatrix} w_{-\frac{p-1}{2},-\frac{q-1}{2}} & \cdots & w_{-\frac{p-1}{2},0} & \cdots & w_{-\frac{p-1}{2},\frac{q-1}{2}} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ w_{0,-\frac{q-1}{2}} & \cdots & w_{0,0} & \cdots & w_{0,\frac{q-1}{2}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ w_{\frac{p-1}{2},-\frac{q-1}{2}} & \cdots & w_{\frac{p-1}{2},0} & \cdots & w_{\frac{p-1}{2},\frac{q-1}{2}} \end{bmatrix} \quad (1)$$

$$\begin{aligned} g(x,y) &= f(x,y) * w(i,j) \\ &= \sum_{i=-\frac{p-1}{2}}^{\frac{p-1}{2}} \sum_{j=-\frac{q-1}{2}}^{\frac{q-1}{2}} f(x+i, y+j) \times w(i,j) \end{aligned} \quad (2)$$

La técnica de convolución permite diversos resultados en las imágenes procesadas de acuerdo con la estructura del núcleo de convolución. Dicha estructura puede ser aprovechada para modificar la arquitectura interna del bloque de procesado, ajustando el consumo de recursos en dependencia de las necesidades de la aplicación. Si se tiene un núcleo de convolución de 2×2 (Fig. 5), como ejemplo, la implementación hardware de la ecuación (2) produce una arquitectura genérica como la mostrada en la Fig. 5, donde se emplean módulos de multiplicación ($Mult_x$) y sumas en cascada ($AddSub_x$).

Si se analizan los diferentes núcleos de convolución es posible encontrar características que permiten la simplificación de la implementación hardware de la versión genérica y con ello la reducción del consumo de recursos. Una de estas características es la simetría. Si se tiene un núcleo de convolución simétrico a la diagonal principal de 2×2 píxeles (Fig. 6), entonces es posible reducir el número de operaciones de multiplicación realizando la suma de los valores de los píxeles correspondientes a los valores simétricos antes de multiplicarlos por el valor del núcleo, como muestra la Fig. 6. La implementación del operador de convolución con un núcleo simétrico reduce el número de multiplicadores en $p \times (q-1)/2$ para el desarrollo de la operación [29].

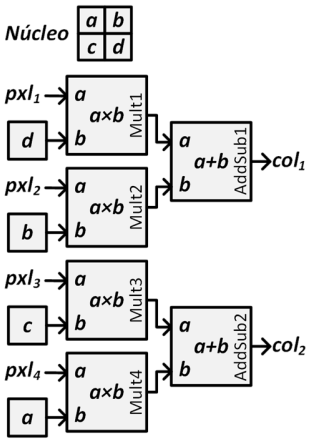


Fig. 5. Arquitectura de la operación de convolución para un núcleo no simétrico.

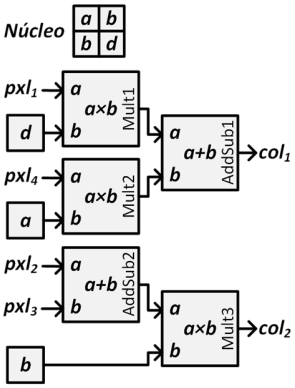


Fig. 6. Arquitectura de la operación de convolución para un núcleo simétrico.

Otra característica de los núcleos de convolución es la presencia de valores reducibles (−1, 0 y 1), opción asociada a la normalización de las operaciones [28], los cuales no precisan del uso de un multiplicador y pueden ser sustituidos o eliminados del diseño contribuyendo a simplificar la arquitectura genérica tal como se muestra en la Fig. 7.

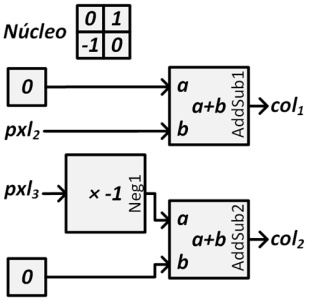


Fig. 7. Arquitectura de la operación de convolución para un núcleo con valores reducibles.

La estructura del bloque de convolución genérico de la biblioteca *XIL XSGImgLib* (*XIL Convolution*) se muestra en la Fig. 8 y su operación es la siguiente.

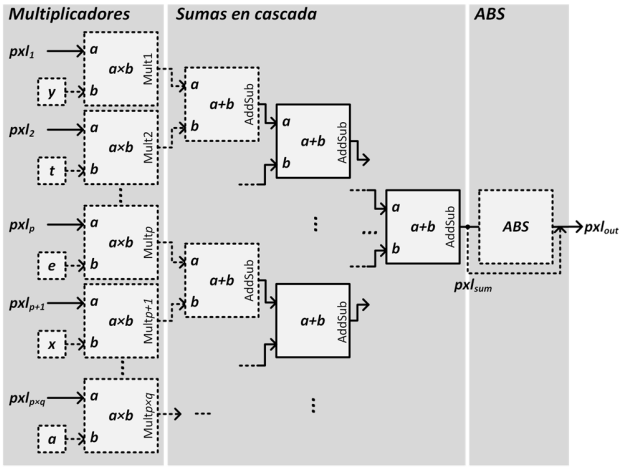


Fig. 8. Arquitectura del bloque de convolución genérico.

Luego de la conversión a paralelo de los píxeles que llegan desde el almacenador de líneas, etapa necesaria para las operaciones de ventana, se realiza la etapa de multiplicación de los $p \times q$ píxeles por el núcleo rotado 180º, enviando los valores del núcleo de convolución a los bloques de multiplicación ($Mult_x$) en conjunto con los píxeles de la imagen en la ventana. La próxima etapa es la suma en cascada del resultado de los bloques de multiplicación, para la cual se emplean módulos de suma (*AddSub*) de dos entradas. Por último, el bloque de cálculo de valor absoluto (*ABS*) efectúa esta operación para el resultado de la suma en cascada.

Para ajustar la arquitectura del bloque a las necesidades de la aplicación que se desarrolla, durante el diseño de la aplicación los valores del núcleo son analizados automáticamente para detectar la simetría y la presencia de valores reducibles en el mismo, ejecutando el procedimiento de modificación automática de la arquitectura del bloque. Para el caso de la simetría, la etapa de multiplicación y la primera etapa de suma (marcadas en líneas discontinuas) son reconfiguradas para reducir la cantidad de bloques de multiplicación, mientras que, para los valores reducibles, los multiplicadores son eliminados o substituidos según se precise.

La ventana de configuración del bloque presenta opciones que permiten generar una nueva arquitectura a partir de la anterior (Fig. 9). Entre estos parámetros se encuentran la selección de la operación (*Operation*) entre los núcleos predefinidos *Edge*, *Sobel*, *Prewitt*, *Laplace*, *Blur*, *Smooth*, *Gaussian*, *Average*, *Sharpen*, *Identity* o introducir una nueva operación manualmente (*Manual Entry*); la normalización del núcleo (*Use normalized kernel*); el análisis de la simetría (*Automatic detect symmetric kernel*); y el uso de valor absoluto (*Absolute value output*).

Una vez configurado el bloque se implementa el diseño empleando las herramientas de Xilinx desde System Generator y se descarga el archivo de configuración en el FPGA. El consumo de recursos del bloque depende del tamaño de la ventana de procesado, de los valores del núcleo de convolución y de las opciones de configuración empleadas. El peor caso se obtiene para la operación de promedio de los píxeles en la ventana (*Average*), con núcleo normalizado y a total precisión, ya que la estructura del núcleo no permite reducir ningún elemento. La TABLA I muestra algunos resultados de implementación sobre un FPGA *Spartan-6 LX45*. La configuración simétrica, en comparación con la no simétrica, reduce los bloques de multiplicación y las tablas de búsquedas (LUT: *Look-Up Table*) en comparación con la configuración simétrica de 5×5. Además, la velocidad de procesado alcanzada en el peor caso es de 281 MHz, lo cual equivale a procesar un pixel cada 3,57 ns.

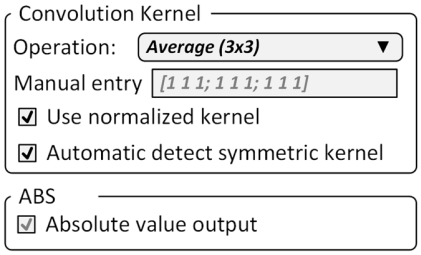


Fig. 9. Segmento de la ventana de configuración del bloque de convolución genérico.

TABLA I
COMPARACIÓN DEL CONSUMO ENTRE DIFERENTES BLOQUES DE CONVOLUCIÓN GENÉRICOS

Recursos (Spartan-6 LX45)		XIL Convolution Average 5×5			Conv. XSG 5×5
		No simétr.	Simétrica		
Tipo	Disponibles	Normalizada		No norm.	
Registros	**54 576**	160	160	160	618
LUT	**27 288**	477	350	318	431
RAMB8	**232**	0	0	0	5
DSP48A	**58**	25	15	0	5
Frecuencia (MHz)		333,000	280,899	375,093	47,619

Los resultados del análisis de la opción de normalización de las operaciones dependen del núcleo de convolución empleado. Para el caso de la operación promedio, el núcleo no normalizado presenta todos los valores en uno, por lo que se aplican las opciones para reducir los multiplicadores, alternativa imposible cuando se normaliza el núcleo. Para una ventana de 5×5, la normalización de las operaciones del bloque de convolución genérico con configuración simétrica aumenta en 32 el consumo de las tablas de búsquedas para el núcleo *Average*. Además, se emplean 25 bloques de multiplicación en la configuración no simétrica normalizada, cantidad reducida al emplear las otras opciones.

Comparando el bloque *XIL Convolution*, en su peor caso, con respecto al bloque equivalente disponible en la biblioteca de Xilinx System Generator (*Conv.XSG 5×5*) en [25], el primero presenta una reducción de un 74,11% en los registros. En cuanto a los multiplicadores, se emplean 25 bloques DSP48A, mientras que el bloque de XSG utiliza cinco bloques de multiplicación y cinco bloques BRAM de ocho bits por su arquitectura semiparalela, la cual reduce considerablemente la frecuencia de trabajo. La máscara de configuración que

presenta el bloque de XSG se limita a la parametrización del núcleo de convolución, sin analizar otras opciones que permitan adaptar la arquitectura a la aplicación.

B. Filtros Específicos Lineales

Los núcleos de convolución utilizan una serie de valores predefinidos para operaciones específicas, por lo que es posible diseñar arquitecturas que reduzcan el consumo de recursos con respecto a la operación genérica. El desarrollo de estos módulos tiene como base la implementación de la definición matemática de la operación de convolución de la ecuación (2), analizando las características del núcleo que permitan simplificar las operaciones. También se considera la presencia de columnas idénticas en la estructura del núcleo de convolución para ser reutilizadas en la etapa de sumas en cascada, como se muestra en la Fig. 10. Estas características son empleadas en la implementación de los filtros específicos de la biblioteca *XIL XSGImgLib*, realizando las operaciones de multiplicación con desplazamientos y sumas para reducir el consumo de recursos del bloque de procesado.

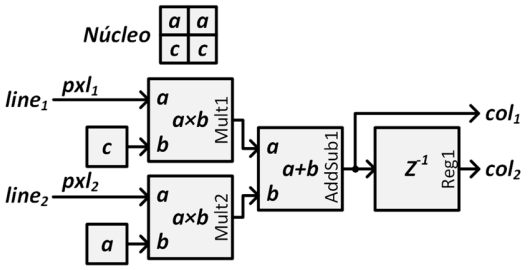


Fig. 10. Arquitectura de la operación de convolución para un núcleo con similitud de columnas.

La Fig. 11 muestra la arquitectura interna básica de los filtros específicos en la biblioteca *XIL XSGImgLib*. La primera etapa es la operación a realizar con los píxeles de la imagen (*Filtro*) y está definida por los valores del núcleo de convolución de cada filtro. Seguidamente se procede a la normalización del resultado y, por último, se obtiene el valor absoluto de la operación (*ABS*). Estas dos últimas etapas pueden estar presentes o no en la arquitectura, según las opciones de configuración seleccionadas por el diseñador y la estructura interna del núcleo. La opción de normalización añade un bloque de multiplicación por una constante o un bloque de desplazamiento entre la salida del filtro (pxl_{sum}) y la señal intermedia pxl_{norm}. En caso contrario, este bloque es eliminado y se conectan las señales anteriores. Si el núcleo no presenta valores negativos la operación no precisa del bloque de valor absoluto, eliminándolo y enlazando la salida de la etapa previa (pxl_{norm}) con la salida del bloque (pxl_{out}).

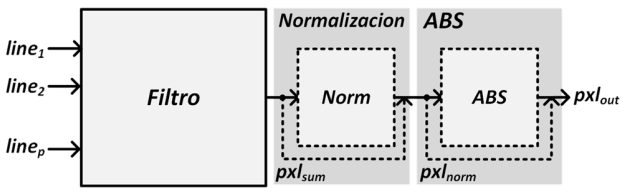


Fig. 11. Arquitectura interna básica de los filtros lineales específicos.

El filtro *Smooth* de *5×5* es unos de los filtros con mayores aportes en la arquitectura entre los filtros específicos, ya que el núcleo correspondiente (Fig. 12) presenta valores reducibles en todo el borde, similitud de columnas entre las columnas 1 - 5 y 2 – 4, y es posible la reutilización de operaciones en la multiplicación por 44.

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

Fig. 12. Núcleo de convolución $Smooth$.

Para la multiplicación por cinco (*×5*) se realiza un desplazamiento de dos bits a la izquierda y la suma de la entrada, mientras que la multiplicación por 44 emplea el operador anterior además de desplazamientos y sumas, como muestra la Fig. 13. El uso de un multiplicador de *×44/5* reduce la cantidad multiplicadores por cinco (*×5*) en la arquitectura, aprovechando la operación del valor central en la segunda columna y obtener un multiplicador por 44 en el centro de la ventana.

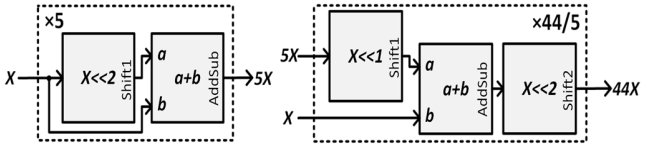


Fig. 13. Módulos de multiplicación empleados en la arquitectura del filtro $Smooth$.

El consumo de recursos del filtro *XIL Smooth* y la comparación con el bloque de convolución genérica (*XIL Convolution*) configurado para la operación $Smooth$ se muestran en la TABLA II. El bloque de convolución genérico presenta un consumo mayor al utilizado por el filtro específico *XIL Smooth* para un FPGA *Spartan-6 LX45* y emplea, además, 15 bloques de multiplicación. En cuanto a la frecuencia de procesado, esta se reduce en 100 MHz para el bloque específico *XIL Smooth* en comparación con el bloque de convolución genérico, aunque el ciclo de operaciones de cada píxel es de 5,515 ns, tiempo suficiente para procesar una imagen de resolución *Full HD* a 75 Hz.

TABLA II
COMPARACIÓN DEL CONSUMO ENTRE EL FILTRO SMOOTH Y EL BLOQUE DE CONVOLUCIÓN GENÉRICO

Recursos (Spartan-6 LX45)		XIL Smooth	XIL Convolution
Tipo	**Disponibles**		**Smooth simétrico**
Registros	**54 576**	104	160
LUT	**27 288**	264	406
DSP48A	**58**	0	15
Frecuencia (MHz)		182,324	282,646

IV. SEGMENTACIÓN DE FOTOGRAMAS CON *XIL XSGIMGLIB*

La unión y configuración de los bloques disponibles en la

biblioteca de procesado de imágenes y vídeos *XIL XSGImgLib* permite el desarrollo de sistemas de procesado complejos, liberando al desarrollador de los detalles de la implementación hardware de los diferentes bloques, y reduciendo el tiempo de diseño de la unidad de procesado. Las aplicaciones de la biblioteca son amplias, incluyendo hasta el momento bloques de procesado de bajo, medio y alto nivel.

Un ejemplo del empleo de la biblioteca en la creación de sistemas complejos es la segmentación de fotogramas, operación que se utiliza para separar los objetos de interés del fondo de una imagen como paso previo para la etapa de extracción de características, usual en aplicaciones de detección de movimiento, conteo de objetos y análisis de estructuras.

A partir de una implementación software equivalente para esta aplicación, el diseñador puede identificar los correspondientes bloques que se necesitan de la biblioteca *XIL XSGImgLib* sin tener necesidad de conocer sus detalles de implementación hardware, acelerando así el proceso de diseño.

La unidad de procesado (Fig. 14) emplea nueve bloques de la biblioteca *XIL XSGImgLib*, realizando la conversión a escala de grises de imágenes *RGB*, a las cuales se les aplican diversas operaciones de filtrado espacial como reducción de ruidos, detección de bordes, umbral, suavizado de los píxeles y operaciones morfológicas. En las etapas de simulación y cosimulación del diseño, la entrada y la salida (*Módulo de Lectura*$_1$ y *Control de Vídeo* respectivamente) son emulados a través de bloques de Simulink®, intercambiando información con el espacio de trabajo de MATLAB® y posibilitando la comprobación software del diseño antes de la implementación física. Una vez comprobado el diseño se implementa el sistema de procesado sobre el hardware del FPGA y se conecta la entrada y la salida al respectivo flujo de vídeo de la aplicación.

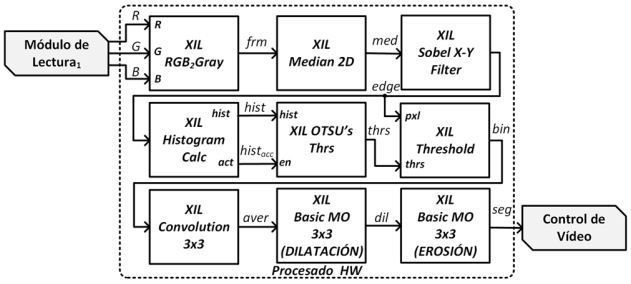


Fig. 14. Unidad de procesado para la segmentación de fotogramas basada en bloques de la biblioteca *XIL XSGImgLib*.

El módulo de conversión (*XIL RGB2Gray*) recibe el fotograma a color (Fig. 15a) del flujo de vídeo a procesar, transformándolo a escala de grises (Fig. 15b). El resultado de la conversión (*frm*) es enviado a la etapa de reducción de ruidos (bloque *XIL Median2D*), empleando para ello un filtro específico de mediana configurado para imágenes de ocho bits de entrada. Esta etapa entrega al módulo de detección de bordes (*XIL Sobel X-Y Filter*) una imagen (*med*) donde el ruido impulsivo se ha eliminado, como muestra la Fig. 15c. Los bordes de los objetos en las imágenes (Fig. 15d) son detectados por medio de un filtro específico tipo *Sobel X-Y* configurado para imágenes de ocho bits a la entrada, salida normalizada y valor absoluto activo.

En la siguiente etapa (Fig. 15e) se ejecutan dos operaciones en paralelo al resultado de la detección de bordes (*edge*): la detección del valor de umbral óptimo (*thrs*) y la obtención del umbral de la imagen (*bin*). El cálculo del valor de umbral óptimo, mediante la aproximación hardware del método de Otsu, depende de la obtención del histograma del fotograma por lo que este valor está listo una vez analizado el fotograma actual y encontrándose el próximo fotograma en la entrada de la unidad de procesado. Teniendo en cuenta que los cambios entre fotogramas consecutivos en una secuencia de vídeo no presentan diferencias significativas, es posible utilizar el resultado del valor del umbral óptimo obtenido para el fotograma actual en el procesado del posterior [30]. Empleando este principio, el resultado del valor de umbral óptimo del fotograma posterior no es muy diferente a obtenido para el fotograma actual, no siendo preciso emplear recursos de almacenamiento para guardar los fotogramas y permitiendo un flujo de procesado continuo.

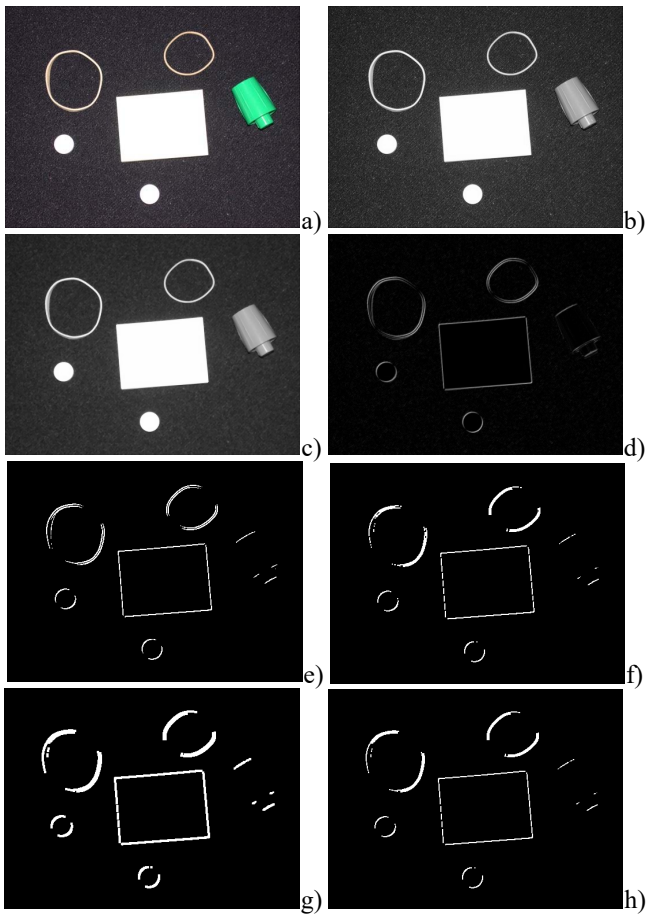


Fig. 15. Resultado de las etapas de segmentado del fotograma. a) Fotograma original en RGB. b) Conversión a escala de grises. c) Reducción de ruidos. d) Detección de bordes. e) Umbral. f) Suavizado. g) Dilatación. h) Segmentación.

La salida del bloque *XIL Threshold* se conecta a un bloque de convolución genérico (*XIL Convolution 3×3*) para reducir el ruido no eliminado en la etapa anterior (Fig. 15f). Este bloque se configura para una operación de promedio de los píxeles, con entrada para imágenes binarias, núcleo normalizado y con

detección de simetría.

Las últimas etapas del proceso de segmentación son realizadas por operadores morfológicos básicos (*XIL Basic MO 3×3*). El primer bloque entrega la dilatación (*dil*) del suavizado obtenido en la etapa anterior (*aver*), agrupando puntos aislados que no fueron eliminados en las operaciones anteriores (Fig. 15g); mientras que el bloque de erosión retorna el resultado de la dilatación a su tamaño original (Fig. 15h), entregando la imagen segmentada (*seg*) a la interfaz de salida.

Los detalles de implementación de la unidad de procesado hardware sobre un FPGA *Spartan-6 LX45* se muestran en la TABLA III. La cantidad de recursos lógicos empleados alcanza el 4,23% de los registros, el 6,27% de las tablas de búsqueda, seis multiplicadores dedicados (10,34%) para el bloque de convolución genérico, y dos módulos de memoria interna de ocho bits (0,86%) utilizados en el bloque de cálculo del histograma. La unidad de procesado opera a una frecuencia máxima de 43,102 MHz, procesando un píxel cada 23,2 ns y permitiendo el trabajo con fotogramas de hasta 800×600 píxeles (SVGA) de resolución a 75 Hz.

TABLA III
CONSUMO DE RECURSOS DE LA UNIDAD DE PROCESADO HARDWARE PARA LA SEGMENTACIÓN

Recursos (Spartan-6 LX45)		**Segmentación**	
Tipo	**Disponibles**	**Ocupados**	**Por ciento**
Registros	**54 576**	2 310	4,23%
LUT	**27 288**	1 712	6,27%
RAMB8	**232**	2	0,86%
DSP48A	**58**	6	10,34%

V. CONCLUSIONES

El flujo de diseño basado en modelos proporcionado por System Generator sobre el entorno MATLAB®/Simulink®, en combinación con las herramientas específicas de desarrollo de FPGA para la implementación de sistemas de procesado hardware, presenta un nivel de abstracción superior al de un flujo de diseño basado en HDL y facilita considerablemente el desarrollo de sistemas de procesado de imágenes y vídeos, liberando al diseñador de los detalles específicos de las implementaciones de cada uno de los componentes. Utilizando este flujo de diseño se han concebido, desarrollado, implementado y validado un total de 54 bloques para 38 operaciones de procesado de imágenes y vídeos, los cuales, en combinación con el procedimiento de modificación automática de la arquitectura, permiten ajustar el consumo de recursos a las necesidades de cada aplicación.

Se han desarrollado arquitecturas genéricas que permiten reducir el consumo de recursos hardware aprovechando características específicas de la configuración del bloque o de su funcionamiento. Por ejemplo, en el bloque de convolución genérico (y otros que lo permitan) se aprovechan las características de simetría, valores reducibles y normalización del núcleo para simplificar su arquitectura. En los filtros específicos se aprovechan las características particulares del núcleo de convolución, como la similitud de columnas o la simetría, y también se sustituyen los bloques de multiplicación por desplazamientos y sumas para reducir el consumo de recursos del FPGA.

Se ha evidenciado la utilidad de la biblioteca *XIL XSGImgLib* en la aceleración del proceso de diseño mediante la implementación de un sistema de procesado de imágenes y vídeos complejo, empleando diversos bloques conectados entre sí. Los resultados obtenidos ponen de manifiesto las ventajas del flujo de diseño basado en modelos de XSG, así como la utilidad de las opciones de configuración de los bloques.

AGRADECIMIENTOS

Esta investigación ha sido financiada parcialmente por la Agencia Española de Cooperación Internacional para el Desarrollo (AECID) y por el Ministerio de Economía y Competitividad español, así como por el programa CAPES/MES mediante el proyecto 219/2013

REFERENCIAS

[1] R. C. González, R. E. Woods, and S. L. Eddins, *Digital Image Processing using MATLAB*, 2nd ed. USA: Gatesmark Publishing, 2009.

[2] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV," *Communications of the ACM*, vol. 55, no. 6, pp. 61–69, 2012.

[3] G. Bradski and A. Kaehler, *Learning OpenCV*, 1st ed. Gravenstein Highway North, Sebastopol, CA: O'Reilly Media, Inc., 2008.

[4] R. C. González and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 2007.

[5] F. Grimm, H. Bunke, and J. Hählen, "An approach to expert systems for image processing software libraries," *Mathematics and Computers in Simulation*, vol. 36, pp. 303–313, 1994.

[6] A. Toledo Moreo, P. Navarro Lorente, F. Soto Valles, J. Suardíaz Muro, and C. Fernández Andrés, "Experiences on developing computer vision hardware algorithms using Xilinx System Generator," *Microprocessors and Microsystems*, vol. 29, pp. 411–419, 2005.

[7] C. Vicente-Chicote, A. Toledo Moreo, and P. Sánchez-Palma, "Image Processing Application Development: From Rapid Prototyping to SW/HW Co-simulation and Automated Code Generation," in *Second Iberian Conference, Pattern Recognition and Image Analysis Lecture Notes in Computer Science*, 2005, pp. 659–666.

[8] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*, 1st ed. Solaris South Tower, Singapore: John Wiley & Sons (Asia) Pte Ltd, 2011.

[9] J. Hiraiwa and H. Amano, "An FPGA Implementation of Reconfigurable Real-Time Vision Architecture," in *27th International Conference on Advanced Information Networking and Applications Workshops*, 2013, pp. 150–155.

[10] M. Gorgon and J. Pryzybylo, "FPGA based controller for heterogenous image processing system," in *Euromicro Symposium on Digital Systems Design*, 2001, pp. 453–457.

[11] M. Gorgon and R. Tadeusiewicz, "Hardware-based image processing library for Virtex FPGA," *Reconfigurable Technology: FPGAs for Computing and Applications II*, vol. 4212, pp. 1–10, 2000.

[12] M. Genovese and E. Napoli, "An FPGA-based Real-time Background Identification Circuit for 1080p Video," in *Eighth International Conference on Signal Image Technology and Internet Based Systems*, 2012, pp. 330–335.

[13] K. Wiatr and E. Jamro, "Implementation image data convolutions operations in FPGA reconfigurable structures for real-time vision systems," in *International Conference on Information Technology: Coding and Computing*, 2000, pp. 152–157.

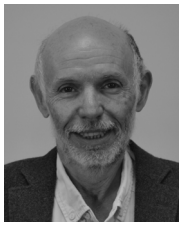
[14] A. Toledo Moreo, S. Cuenca-Asensi, and J. Suardíaz Muro, "Codesign Environment for Computer Vision Hw/Sw Systems," in *5th International Workshop on Informational Optics*, 2006, vol. 860, no. 1, pp. 527–536.

[15] L. M. Garcés-Socarrás, A. J. Cabrera Sarmiento, S. Sánchez-Solano, P. Brox Jiménez, E. Ieno, and T. C. Pimenta, "Modificación automática de arquitecturas de módulos hardware de procesado de imágenes," *Revista de Ingeniería Electrónica, Automática y Comunicaciones*, vol. XXXVII,

no. 3/2016, pp. 21–33, 2016.
[16] J. Maddocks and R. Williams, "VHDL Image Processing Source Modules," Hunt Engineering, Brent Knoll, Somerset, UK, 2006.
[17] J. Maddocks and R. Williams, "Image Processing VHDL for FPGA modules," *Hunt Engineering*, 2006. [Online]. Available: http://www.hunteng.co.uk/products/ip/imaging_ip.htm. [Accessed: 22-Jan-2015].
[18] V. Carrier, G. Gourat, and C. Russo, "Image Processing Library & algorithms porting support to FPGA," *NEXVISION eyetronic*, 2014. [Online]. Available: http://nexvision.fr/solutions/reference-design-products/pixtera/. [Accessed: 21-Jan-2015].
[19] Xilinx, "Xilinx CORE Generator System," 2014. [Online]. Available: http://www.xilinx.com/tools/coregen.htm. [Accessed: 21-Jan-2015].
[20] K. Rupnow, "A study of high-level synthesis: Promises and challenges," in *9th IEEE International Conference on ASIC*, 2011, pp. 1102–1105.
[21] D. G. Bailey, "The Advantages and Limitations of High Level Synthesis for FPGA Based Image Processing," in *9th International Conference on Distributed Smart Cameras*, 2015, pp. 134–139.
[22] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong, "AutoPilot: A Platform-Based ESL Synthesis System," in *High-Level Synthesis from Algorithm to Digital Circuit*, P. Coussy and A. Morawiec, Eds. Dordrecht: Springer Netherlands, 2008, pp. 99–112.
[23] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate area and delay estimators for FPGAs," in *Design, Automation and Test in Europe*, 2002, pp. 862–869.
[24] L. M. Garcés-Socarrás, S. Sánchez-Solano, P. Brox Jiménez, and A. J. Cabrera Sarmiento, "Library for model-based design of image processing algorithms on FPGAs," *Revista de la Facultad de Ingeniería Universidad Antioquia*, vol. 1, no. 68, pp. 36–47, 2013.
[25] Xilinx, *System Generator for DSP Referece Guide*, vol. 638, no. 14.5. San José, CA, USA: Xilinx Inc., 2013.
[26] Xilinx, *System Generator for DSP Getting Started Guide*, vol. 639, no. 14.3. San José, CA, USA: Xilinx Inc., 2012.
[27] Xilinx, *System Generator for DSP User Guide*, vol. 640, no. 14.3. San José, CA, USA: Xilinx Inc., 2012.
[28] L. M. Garcés-Socarrás, "Desarrollo de una biblioteca basada en modelos de módulos hardware configurables para procesado de imágenes y vídeos," Tesis Doctoral, Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), La Habana, 2016.
[29] R. Turney, "Two-Dimensional Linear Filtering," *Xilinx Application Notes*, vol. 933, no. 1.1, pp. 1–8, 2007.
[30] U. Bidarte, J. L. Martin, A. Zuloaga, and J. Ezquerra, "Adaptive image brightness and contrast enhancement circuit for real-time vision systems," in *IEEE International Conference on Industrial Technology*, 2000, pp. 421–426.

Luis Manuel Garcés Socarrás, Ingeniero en Automática en 2006, Máster en Sistemas Digitales en 2011 y Doctor en Ciencias Técnicas en 2017 por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Titular del Departamento de Automática y Computación de la CUJAE, Calle 114 Nº 11901 e/ Ciclo vía y Rotonda, CUJAE, Marianao, La Habana, Cuba. Telef: +53 7 266 3341. Actualmente desarrolla el tema de investigación de aceleración de algoritmos mediante hardware reconfigurable para el procesado de imágenes y vídeos. Email: lmgarcess@automatica.cujae.edu.cu.

Alejandro José Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Titular del Departamento de Automática y Computación de la CUJAE, Calle 114 Nº 11901 e/ Ciclo vía y Rotonda, CUJAE, Marianao, La Habana, Cuba. Telef: +53 7 266 3301. Sus líneas de investigación principales están relacionadas con el desarrollo de sistemas empotrados basados en FPGA y la aceleración de algoritmos sobre hardware reconfigurable, con aplicaciones en procesado de imágenes, control inteligente y criptografía. Email: alex@automatica.cujae.edu.cu.



Santiago Sánchez Solano, Doctor en Ciencias Físicas por la Universidad de Sevilla en 1990, Investigador Científico del CSIC adscrito al Instituto de Microelectrónica de Sevilla, IMSE-CNM, (CSIC/Universidad de Sevilla), Sevilla, España. Sus líneas de investigación se centran en el desarrollo de sistemas empotrados con componentes hardware/software sobre FPGA y la realización microelectrónica de sistemas neurodifusos, así como sus aplicaciones en robótica, procesamiento de imágenes, seguridad y redes de sensores inteligentes. Email: 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 adscrito al Instituto de Microelectrónica de Sevilla, IMSE-CNM, (CSIC/Universidad de Sevilla), Sevilla, España. Email: brox@imse-cnm.csic.es.

Egídio Ieno Junior, Graduado de Ingeniería Eléctrica en 2001, Máster en Telecomunicaciones en 2003 por el Instituto Nacional de Telecomunicaciones (INATEL), Santa Rita do Sapucaí, Minas Gerais, Brasil. Actualmente es profesor del Centro Federal de Educación Tecnológica de Minas Gerais (CEFET MG). Realiza su investigación doctoral en algoritmos de detección de movimiento sobre hardware reconfigurable en la Universidad Federal de Itajubá (UNIFEI) en colaboración con la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE). Email: egidio_ieno@yahoo.com.br.

Tales Cleber Pimenta, Graduado de Ingeniería Eléctrica en 1985, Máster en Ingeniería Eléctrica por la Universidad Federal de Itajubá (UNIFEI), Itajubá, Minas Gerais, Brasil. Doctor en Ingeniería Eléctrica y Computación por la Universidad de Ohio, Estados Unidos. Realizó estudios Postdoctorales en universidades de Estados Unidos. Es profesor de la Universidad Federal de Itajubá desde 1985 en las áreas de circuitos digitales y circuitos integrados. Actualmente desarrolla investigaciones en áreas de microelectrónica y circuitos integrados para aplicaciones biomédicas. Email: tales@unifei.edu.br.