RIELAC, Vol. XXXVII 3/2016 p. 21-33 Septiembre - Diciembre ISSN: 1815-592

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Modificación automática de arquitecturas de módulos hardware de procesado de imágenes

Luis M. Garcés-Socarrás, Alejandro J. Cabrera Sarmiento, Santiago Sánchez-Solano, Piedad Brox Jiménez, Egidio Ieno Junior, Tales Cleber Pimenta

RESUMEN / *ABSTRACT*

El presente artículo describe el empleo del flujo de diseño basado en modelos para el desarrollo de bloques reconfigurables automáticamente para el procesado de imágenes sobre FPGA. Para ello se han concebido arquitecturas hardware que aprovechan características específicas de algunos algoritmos de procesado y que pueden ser modificadas a través de un novedoso procedimiento software. Este aspecto, unido a las restantes opciones de parametrización de los diferentes módulos, permite liberar al diseñador de los detalles específicos de las implementaciones hardware así como adaptar el consumo de recursos del FPGA a las necesidades de la aplicación. El proceso de reconfiguración automática se ilustra con el bloque de convolución genérico realizando comparaciones entre implementaciones de diferentes arquitecturas sobre un FPGA Spartan-6 LX45.

Palabras claves: flujo de diseño basado en modelos, Xilinx System Generator, reconfiguración automática, *XIL XSGImgLib*

This paper describes the use of model-based design flow in the development of self-configurable image processing blocks over FPGAs. For this, some generic hardware architectures has been created and modified using a novel software proceeding using the specific characteristics of the different processing algorithms. This aspect, in addition to others configuration options of the blocks, allows the adjustment the FPGA resources consumption for a specific application. Also the automatic reconfiguration process is analyzed for a generic convolution block making some architectural comparisons over a Spartan-6 LX45 FPGA.

Key words: model-base deseing flow, Xilinx System Generation, self-configuration, XIL XSGImgLib

Automatic architecture modification of hardware modules for image processing

1. -INTRODUCCIÓN

La visión computacional es una transformación de los datos de una imagen o vídeo para generar una nueva representación o decisión. En un sistema de visión computacional, la unidad de procesado recibe una serie de datos numéricos desde el elemento de adquisición o almacenamiento realizando el análisis de las imágenes, así como el control del elemento de captura. Estas operaciones se definen por medio de algoritmos que ejecutan técnicas conocidas como procesado digital de imágenes o vídeo dentro de la unidad de procesado, cuyo objetivo es convertir la secuencia de datos numéricos en una percepción visual y detectar patrones de interés en una imagen o secuencia para la toma de decisiones del sistema al que pertenece [1–5].

La unidad de procesamiento es la parte más importante del sistema de visión computacional la cual se puede implementar sobre computadoras personales (PC), dispositivos secuenciales empotrados o dispositivos hardware paralelos. El consumo de potencia de una PC, en conjunto con el tamaño, el peso y el costo que pueden alcanzar, descartan a esta variante como una solución viable para aplicaciones portables y autónomas, como los sistemas de cámaras inteligentes y vídeo vigilancia automatizada. Las soluciones secuenciales empotradas presentan limitaciones en cuanto a la velocidad de procesado en comparación con los procesadores disponibles en las PC por lo que, cuando la velocidad sea un parámetro crítico, se hace

necesario el empleo de dispositivos hardware paralelos, como los circuitos integrados de aplicaciones específicas (ASIC) y los arreglos de compuertas lógicas programables (FPGA), para implementar las unidades de procesado de imágenes y vídeos [6–12].

La implementación de algoritmos de visión computacional sobre dispositivos hardware paralelos es una tarea ardua y compleja, donde es preciso un dominio sobre la plataforma de implementación y las herramientas de desarrollo. Una forma de solucionar 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, 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 [2,3,13].

El diseño de hardware digital basado en modelos es un nivel de abstracción intermedio que proporciona una estrategia de desarrollo de sistemas empotrados que permite la fácil reutilización de componentes en ambientes de desarrollo gráficos [14]. Este presenta la ventaja de ser más intuitivo para el desarrollo de algunas aplicaciones, entre ellas el procesado de imágenes y vídeos, con respecto a otras técnicas de diseño, ya que posibilita la creación de un prototipo con un comportamiento igual a como lo hacen en el hardware, permitiendo además la simulación con condiciones más reales y complejas. De esta forma la implementación de algoritmos sobre hardware se logra con una alta abstracción permitiendo la síntesis de componentes en hardware [5,12,13,15,16].

Para la implementación de algoritmos de visión computacional es conveniente la disponibilidad de bibliotecas que incorporen la mayor cantidad posible de funciones para el procesado. Aunque existe una gran cantidad de bibliotecas de procesado de imágenes y vídeos desarrolladas para plataformas software basadas en PC y para dispositivos empotrados, existen serias limitaciones para la utilización de las bibliotecas para plataformas hardware. Algunas vienen asociadas a dispositivos hardware específicos o al pago de una licencia adicional; otras no se encuentran disponibles o presentan muy pocos componentes, lo que limita su aplicabilidad [1,17–24].

Un aspecto a considerar en las bibliotecas de componentes para procesado de imágenes y vídeos en la disponibilidad de opciones de configuración de los mismos. Las bibliotecas software no poseen grandes opciones de configuración, pues dada la gran capacidad de memoria y la alta velocidad de los procesadores de las computadoras personales actuales, no necesitan de parámetros para optimizar el consumo de recursos de la plataforma de implementación [1,2,17,18]. En cambio, en las bibliotecas para sistemas de procesado hardware las opciones de configuración de sus componentes es un aspecto muy importante para obtener las operaciones deseadas con un adecuado nivel de rendimiento y consumo de recursos en correspondencia con la complejidad de la aplicación. Las opciones de configuración de los bloques disponibles en las bibliotecas hardware existentes son limitadas y dependen de las operaciones a realizar, agrupándose en bloques para operaciones específicas poco versátiles o bloques genéricos poco adaptables.

XIL XSGImgLib es una biblioteca de módulos hardware que ha sido desarrollada para la creación de sistemas de procesado de imágenes y vídeos complejos sobre FPGA utilizando el flujo de diseño basado en modelos de MATLAB®/Simulink®/Xilinx System Generator [14,25]. La misma cuenta con una amplia gama de bloques de procesado que explotan las opciones de configuración para adaptar la arquitectura interna de los mismos a las necesidades de la aplicación, reduciendo así el consumo de recursos hardware de la implementación y liberando al diseñador del conocimiento de los detalles de hardware de la plataforma y la estructura de los diferentes bloques.

En este artículo se describe la reconfiguración automática de la arquitectura de algunos de los módulos hardware disponibles en la biblioteca *XIL XSGImgLib*, aprovechando características específicas de la arquitectura de estos módulos y empleando un novedoso procedimiento de modificación automática del hardware sobre el flujo de diseño basado en modelos, el cual puede ser extendido a otras aplicaciones. La organización del artículo es la siguiente. Primeramente se expone el procedimiento de reconfiguración automática del hardware desarrollado sobre Xilinx System Generator. A continuación se exponen algunas características de los bloques que implementan la operación de convolución que permiten modificar su arquitectura, así como la utilización del procedimiento de reconfiguración automática de la arquitectura de estos bloques, mostrando la influencia de las opciones de configuración sobre el consumo de recursos sobre un FPGA. Finalmente se resumen las conclusiones del trabajo.

2. -PROCEDIMIENTO DE RECONFIGURACIÓN AUTOMÁTICA

Dos de las principales ventajas del flujo de diseño basado en modelos radican en el rápido desarrollo de prototipos de sistemas de procesado y la posibilidad de emplear una máscara de configuración que permite cambiar los parámetros y, en algunos casos, la arquitectura hardware de los bloques de procesado. Para esto último se requiere de un procedimiento (implementado mediante segmentos de código en MATLAB®) que permita ajustar la estructura hardware de los diferentes bloques, según

las opciones de la máscara de configuración, para adecuar la arquitectura a los requerimientos específicos de cada aplicación, manteniendo así una adecuada relación entre el consumo de recursos del FPGA y los requerimientos temporales. Estas opciones aumentan la versatilidad de los bloques de procesado de la biblioteca *XIL XSGImgLib* y permiten el desarrollo de sistemas de visión computacional personalizados.

En el flujo de diseño convencional utilizando lenguajes de descripción de hardware (HDL) la configuración de un módulo de procesado se lleva a cabo mediante parámetros genéricos definidos en el código HDL. Mediante estos parámetros se puede ajustar la cantidad de bits de las entradas y salidas del bloque, así como cambiar la arquitectura interna del mismo, instanciando componentes descritos en el código e interconectándolos entre sí. Para el flujo de diseño basado en modelos la configuración se realiza a partir de la ventana de configuración del bloque principal, también conocida como máscara, parametrizando los módulos que lo componen y permitiendo el cambio de la arquitectura interna del módulo en caso de ser posible. Para esta tarea se requiere de un procedimiento software que la realice.

De forma general, el diseño de un bloque de procesado empleando Simulink® se efectúa mediante la inserción y la unión, por el diseñador, de bloques o módulos disponibles en las bibliotecas de la herramienta, generando un modelo almacenado en un archivo (*Archivo.mdl/slx*). Este proceso (Figura 1) se realiza de forma gráfica sobre un área cuyo punto superior izquierdo representa el origen y donde cada módulo que integra el sistema de procesado se identifica con un nombre ($Módulo_x$) y un tipo ($Tipo_x$) situado en una posición conocida, la cual se especifica por las coordenadas de la esquina superior izquierda (*X1*,*Y1*) y la esquina inferior derecha (*X2*,*Y2*). A esta información se le añaden las interconexiones de las entradas y salidas del módulo con los restantes componentes del diseño, permitiendo realizar un esbozo de la arquitectura del sistema de procesado mediante un segmento de código en MATLAB®. Este diseño es empaquetado en un subsistema de Simulink®, creando un bloque de procesado.

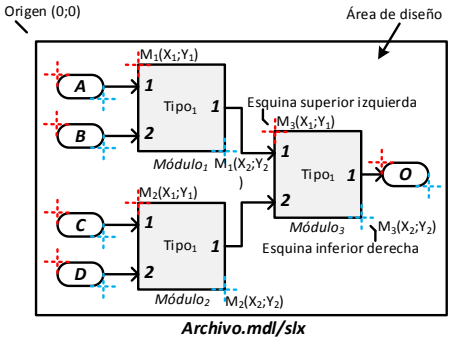

Figura 1

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

MATLAB® provee una serie de funciones que permiten obtener y establecer los parámetros de los componentes de un diseño, así como desconectar, eliminar, conectar y agregar diferentes módulos disponibles en una biblioteca [26–29]. Estas funciones son empleadas en la creación de segmentos de código o subrutinas, incluidos en las funcionalidades de la biblioteca de procesado *XIL XSGImgLib*, para adquirir la posición y la orientación de los módulos del bloque de procesado para la arquitectura genérica del mismo, donde están presentes todos los posibles componentes del diseño. La Figura 2 muestra el proceso de análisis de la arquitectura del nivel superior de un bloque de procesado. El archivo del modelo (*Archivo.mdl/slx*) es explorado por un segmento de código desarrollado en el marco de esta investigación (*Script de análisis.m*) para crear un registro en el espacio de trabajo de MATLAB® (*Bloque.mat*) con los datos que definen a cada módulo en la arquitectura genérica.

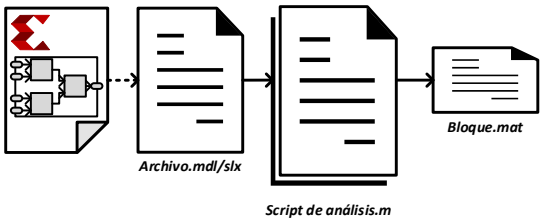

Figura 2

Proceso de análisis de la arquitectura de los bloques de procesado.

La Figura 3 muestra el diagrama de flujo para realizar el análisis de la arquitectura genérica de un bloque de procesado. Todos los módulos (n) que componen el diseño son analizados empleando como referencia sus nombres ($módulo_i$) e indagando si están presentes en la arquitectura. Para los módulos presentes, se obtienen las coordenadas que los delimitan así como su orientación, almacenando esta información en variables del espacio de trabajo. Una vez terminado de procesar todos los módulos, esta información es exportada a un archivo de variables de MATLAB® (*.mat*) para ser empleada en la modificación de la arquitectura.

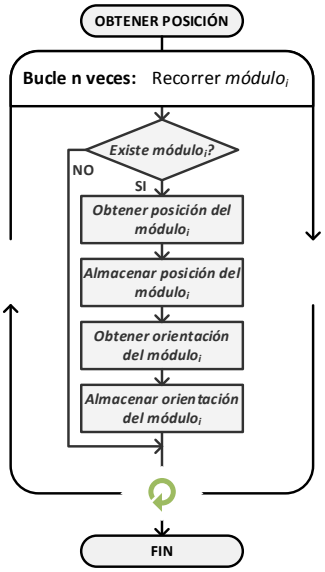

Figura 3 Diagrama de flujo para la obtención de la posición y la orientación de los módulos de un bloque.

El proceso de reconfiguración automática de la estructura interna de un bloque (Figura 4) necesita de un código en MATLAB® (*Script de configuración.m*) que simplifique el trabajo del usuario de la biblioteca a la hora de adaptar la arquitectura a las necesidades de la aplicación. Como parte de las funcionalidades de las bibliotecas de procesado se desarrollan varios segmentos de código en MATLAB® que permiten realizar el cambio de la arquitectura de algunos de los bloques disponibles. Al cambiar las opciones de la máscara el script de configuración, en conjunto con los parámetros de la arquitectura genérica almacenados previamente (*Bloque.mat*), modifica automáticamente el archivo del modelo original creando un nuevo archivo modificado (*Archivo MOD.mdl/slx*), cambiando así la arquitectura interna del bloque desarrollado [29–33].

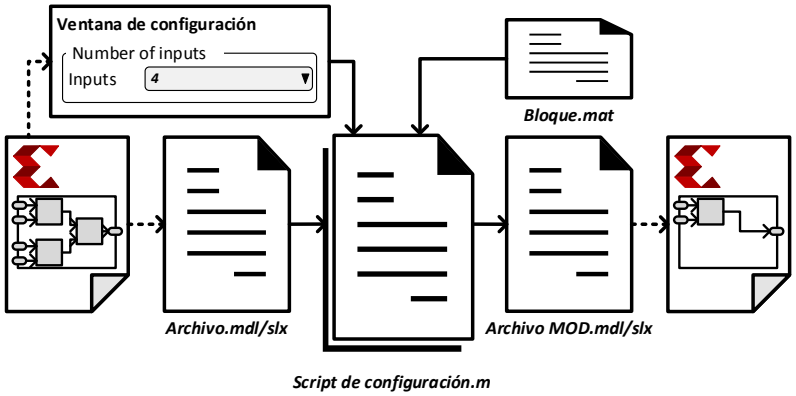

Figura 4
Proceso de reconfiguración automática de los bloques de procesado.

La Figura 1 muestra la arquitectura interna genérica de un bloque configurable que realiza una operación con cuatro valores a la entrada mediante tres módulos de dos entradas cada uno. Al seleccionar una configuración de dos entradas desde la máscara de configuración, se ejecuta automáticamente el script de configuración para realizar el ajuste de la arquitectura interna del bloque de procesado (Figura 5a). Primeramente se detectan los módulos que ya no son necesarios en el diseño (las entradas *C* y *D*; y los módulos *2* y *3*) y se ejecutan las subrutinas para eliminar los mismos.

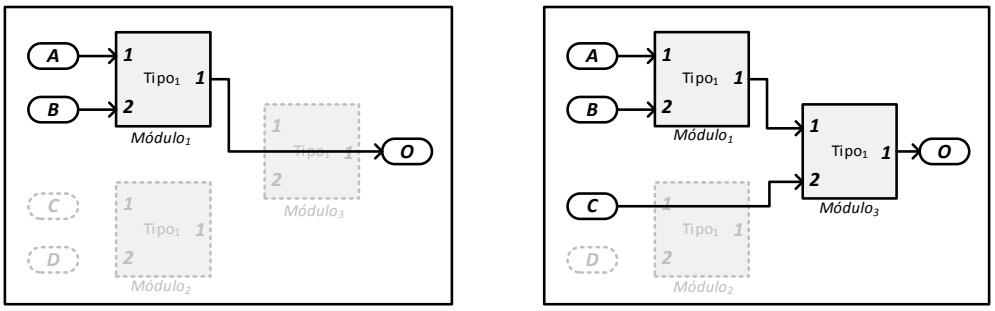

Figura 5

Cambio en la arquitectura interna del bloque de procesado. a) Arquitectura para dos entradas.

Para eliminar un módulo del diseño (*Módulo₃*), como se describe en el diagrama de flujo de la Figura 6a, primeramente se comprueba si el módulo está presente y, en caso afirmativo, se desconectan las entradas y salidas del módulo en análisis (las interconexiones con la salida *O* y los módulos *1* y *2*) para luego eliminarlo. Por último, se interconectan las entradas y salidas de los módulos que han quedado desconectados en el diseño (el *Módulo₁* y la salida *O*), y se ejecuta el mismo algoritmo para eliminar los restantes módulos en la arquitectura (el *Módulo₂* y las entradas *C* y *D*).

En caso de pasar de la arquitectura de dos entradas a una de tres entradas (Figura 5b) el bloque debe ser configurado para esta nueva estructura. Primeramente se analizan los módulos que son necesarios en la nueva arquitectura y cuáles de ellos no se encuentran en la arquitectura actual (la entrada *C* y el *Módulo₃*). Para añadir un módulo se ejecuta un código en MATLAB®, el cual se representa en el diagrama de flujo de la Figura 6b, donde, si el módulo no está presente en la arquitectura (*Módulo₃*), se desconectan los módulos intermedios (el *Módulo₁* y la salida *O*), se añade el módulo correspondiente y se interconecta con los restantes. Por último, se realiza la configuración del módulo añadido, paso también llevado a cabo cuando el módulo existe en el diseño, y se ejecuta la subrutina para el próximo módulo necesario en la arquitectura (la entrada *C*). En la etapa de agregar un módulo al diseño se deben conocer el nombre y su ubicación en las paletas de las bibliotecas incluidas en Simulink®, además de los valores de la posición y la orientación del módulo, adquiridos en el proceso de análisis de la arquitectura genérica.

Otra operación a realizar en la arquitectura de un bloque es la sustitución de un módulo por otro diferente empleando el diagrama de flujo de la Figura 6c. Para esto se ejecutan las subrutinas ya descritas para eliminar y añadir bloques.

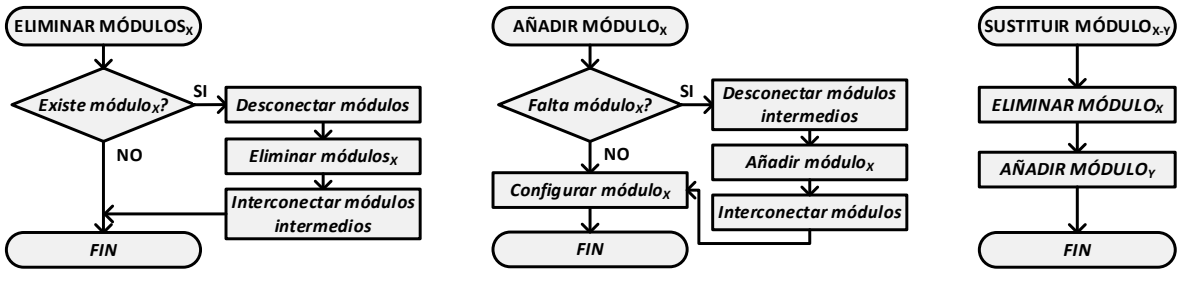

Figura 6

Diagrama de flujo para el cambio de arquitectura de los bloques de procesado. a) Eliminar un módulo existente. b) Añadir un nuevo módulo. c) Sustituir un módulo del diseño.

3.- RECONFIGURACIÓN AUTOMÁTICA DE BLOQUES DE CONVOLUCIÓN

Con el objetivo de ilustrar el procedimiento de reconfiguración automática implementado en la biblioteca *XIL XSGImgLib* a continuación se exponen las características del bloque genérico de convolución que permiten la modificación automática de su arquitectura.

El filtrado espacial es la operación característica del procesado de imágenes, realizando una transformación, lineal o no lineal, sobre los $p \times q$ píxeles de la imagen contenidos en una ventana de p filas y q columnas. Muchas de las etapas de un sistema de procesado de imágenes y vídeos se basan en el filtrado espacial de la imagen $f(x, y)$ con una matriz de vecindad $w(i, j)$, mostrada en la ecuación (1), con centro en la coordenada $(0,0)$ que define las modificaciones a realizar en la imagen (34). De acuerdo a la forma en que viajan los píxeles por el sistema de procesado estos algoritmos necesitan de bloques de conversión del flujo serie de píxeles a paralelo para lograr obtener todos los píxeles en la ventana al mismo tiempo y realizar las operaciones en paralelo.

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

La operación de convolución ($*$) es el exponente principal del procesado espacial lineal de imágenes, donde se desplaza la vecindad de píxeles o núcleo de convolución (w) sobre la imagen (f) evaluando la expresión (2) [34].

$$g(x,y) = f(x,y) * w(i,j) = \sum_{i=-(p-1)/2}^{(p-1)/2} \sum_{j=-(q-1)/2}^{(q-1)/2} f(x+i, y+j) \times w(i,j) \tag{2}$$

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 ajustar la arquitectura interna del bloque de procesado, ajustando el consumo de recursos en dependencia de las necesidades de la aplicación.

Para ilustrar las operaciones a realizar en la convolución de una imagen, si se tiene un núcleo de convolución de 2×2 con valores mostrados en (3), el valor de un píxel resultante de la operación de convolución (2) se obtiene mediante la expresión (4).

$$w(i,j) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{3}$$

$$\begin{aligned} g(x,y) &= f(x-1,y-1) \times w(1,1) + f(x-1,y) \times w(1,0) + f(x,y-1) \times w(0,1) + f(x,y) \times w(0,0) \\ &= d \times f(x-1,y-1) + c \times f(x-1,y) + b \times f(x,y-1) + a \times f(x,y) \end{aligned} \tag{4}$$

La implementación básica de esta operación de convolución (Figura 7a) precisa de módulos de multiplicación (*Mult*$_x$), además de sumas en cascadas (*AddSub*$_x$) para obtener el resultado de un nuevo píxel. En esta figura se muestra la etapa de multiplicación de la operación y la primera etapa de las sumas en cascadas.

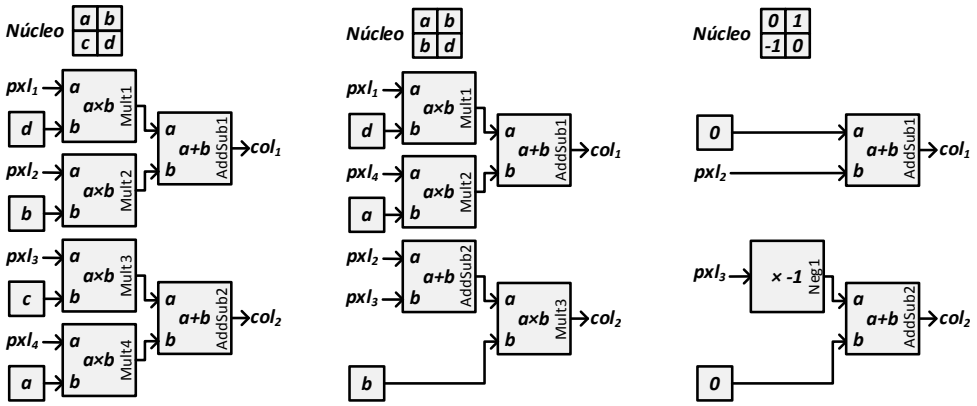

Figura 7

Características de los núcleos de convolución que permiten simplificar la implementación de la operación. a) Núcleo no simétrico. b) Núcleo simétrico. c) Núcleo con valores reducibles.

Si se analiza el núcleo de convolución es posible encontrar características que permiten el cambio de la arquitectura del bloque y conducen a la simplificación de la implementación 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 de 2×2 píxeles, como el mostrado en (5), 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 expresión (6).

$$w(i,j)=\begin{bmatrix} a & b \\ b & d \end{bmatrix} \tag{5}$$

$$\begin{aligned} g(x,y) &= d\times f(x-1,y-1)+b\times f(x-1,y)+b\times f(x,y-1)+a\times f(x,y) \\ &= d\times f(x-1,y-1)+b\times[f(x-1,y)+f(x,y-1)]+a\times f(x,y) \end{aligned} \tag{6}$$

La implementación del operador de convolución con un núcleo simétrico (Figura 7b) reduce el número de multiplicadores en $p(q-1)/2$ necesarios para el desarrollo de la operación definida en la ecuación (6) [35].

Muchos de los núcleos de convolución definidos para las operaciones de modificación de las imágenes presentan valores reducibles en su contenido. Un valor reducible es un número (-1, 0, y 1) que no precisa del uso de un multiplicador el cual puede ser omitido o sustituido por un módulo que emplee menor cantidad de recursos lógicos. Si se tiene un núcleo de convolución de 2×2 píxeles con valores reducibles como el mostrado en (7) entonces es posible simplificar el número de operaciones de multiplicación cuyos valores en el núcleo sean cero, uno y menos uno, como muestra la ecuación (8).

$$w(i,j)=\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{7}$$

$$\begin{aligned} g(x,y) &= 0\times f(x-1,y-1)-1\times f(x-1,y)+1\times f(x,y-1)+0\times f(x,y) \\ &= f(x,y-1)-f(x-1,y) \end{aligned} \tag{8}$$

La implementación de la operación de convolución con términos reducibles (Figura 7c) permite eliminar las entradas cuyo valor en el núcleo sea cero, eliminar el bloque de multiplicación en las entradas con valor unitario y sustituir el multiplicador por el bloque de negado a los píxeles con modificador menos uno.

Otro aspecto importante en el procesado de imágenes y vídeos es la normalización de las operaciones matemáticas. Al trabajar con imágenes en modelos de representación definidos, como RGB o escala de grises, se espera que las imágenes resultantes mantengan la escala original, operación conocida como normalización. Si la salida no presenta la misma escala que la entrada ocurre un desbordamiento en los píxeles de la imagen resultante, aplicándose, según la parametrización del bloque de procesado, el control del desbordamiento de los píxeles. La forma de realizar la normalización del núcleo de convolución depende de la estructura del mismo, dividiendo cada uno de los píxeles por la suma total de sus valores, excepto cuando la

misma es cero, donde se normalizan los valores positivos y negativos del núcleo por separado. Para el caso de la operación promedio, el núcleo no normalizado presenta todos los valores en uno (Figura 8a) siendo estos valores enteros, mientras que el núcleo normalizado presenta valores en el dominio de los números reales (Figura 8b).

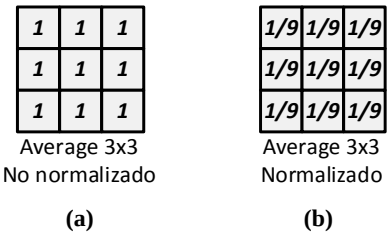

Figura 8

Ejemplo de normalización del núcleo de convolución. a) Núcleo no normalizado. b) Núcleo normalizado.

La implementación del algoritmo de convolución para hardware reconfigurable ha encontrado aplicación práctica en diferentes áreas, siendo una necesidad común la rapidez en la ejecución de los mismos. La arquitectura del bloque genérico de convolución *XIL Convolution*, incluido en la biblioteca *XIL XSGImgLib* y mostrado en la Figura 9, explota las características de simetría, valores reducibles y normalización del núcleo, permitiendo modificar automáticamente su arquitectura en función de la máscara de configuración del bloque. En líneas discontinuas se muestran los bloques y conexiones que pueden ser modificados en la arquitectura dependiendo de las opciones de configuración.

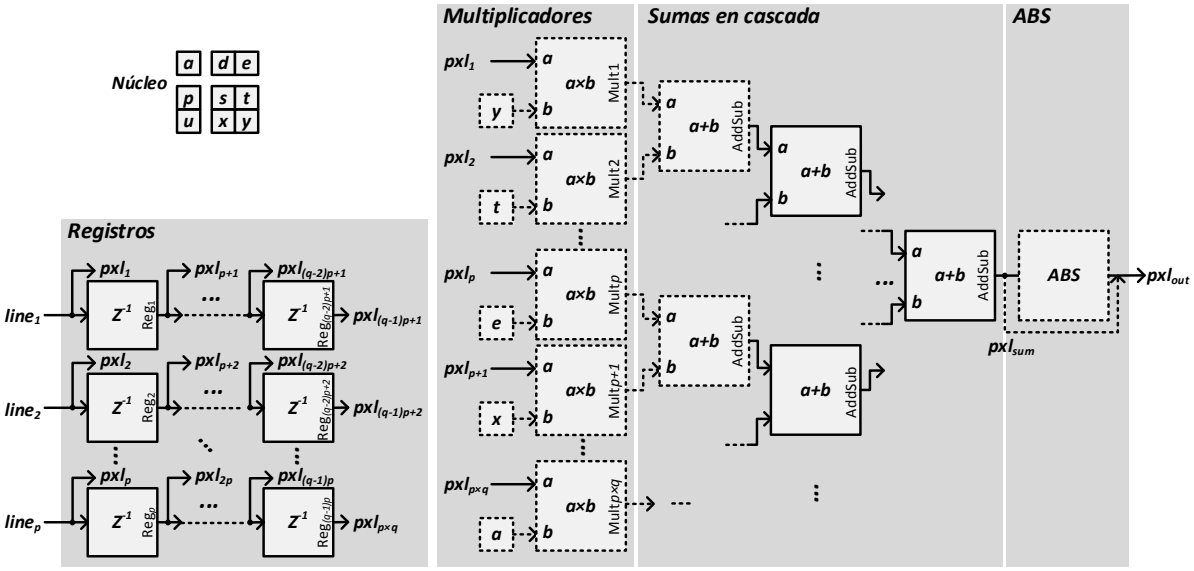

Figura 9

Arquitectura interna del bloque *XIL Convolution*.

La primera etapa del filtrado es la conversión del flujo serie de píxeles a paralelo mediante los bloques de registros. Los píxeles que llegan desde el almacenador de línea ($line_x$) son paralelizados obteniendo $p \times q$ píxeles (pxl_x) a la entrada de la etapa de multiplicación. La segunda etapa de procesado se encarga de la multiplicación de los píxeles por el núcleo rotado 180º. Los valores del núcleo de convolución son enviados a los bloques de multiplicación ($Mult_x$) en conjunto con los píxeles en la ventana a los que modifican. A su vez, los valores del núcleo son analizados para detectar la simetría del mismo y la presencia de valores reducibles para ejecutar el procedimiento de modificación automática de la arquitectura del bloque. Para el caso de la simetría, la etapa de multiplicación en conjunto con la primera etapa de suma 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 tercera etapa del bloque de convolución genérico es la suma en cascada del resultado de los bloques de multiplicación, para la cual se utilizan 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. Este bloque se implementa si el núcleo de convolución presenta valores negativos, colocando el bloque de *ABS* entre la salida de la etapa de suma (pxl_{sum}) y la salida del bloque de procesado (pxl_{out}); en caso contrario el bloque de *ABS* es eliminado, interconectando la etapa de suma con la salida.

La Figura 10 muestra un segmento de la ventana de configuración del bloque de procesado. La parametrización del núcleo de convolución (*Convolution Kernel*) permite la selección de la operación (*Operation*) entre varias ya predefinidas o la introducción de un núcleo manualmente (*Manual entry*). Además, es posible habilitar la detección de simetría en el núcleo seleccionado (*Automatic detect symmetric kernel*), la normalización del núcleo de convolución (*Use normalized kernel*) y el uso de valor absoluto a la salida (*Absolute value output*). Todas estas opciones realizan el análisis del núcleo de convolución y provocan la modificación de la arquitectura empleando el procedimiento descrito en el apartado anterior.

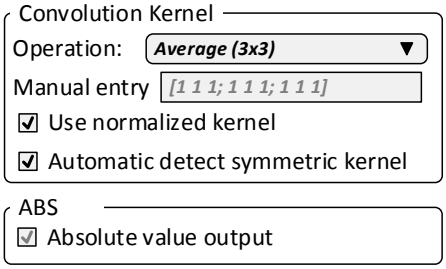

Figura 10

Máscara de configuración del bloque *XIL Convolution*.

El consumo de recursos del bloque de convolución genérico *XIL Convolution* depende del tamaño de la ventana de procesado, la composición 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 total precisión y valores del núcleo normalizados, ya que la composición del núcleo no reduce ningún elemento, utilizando todos los bloques de multiplicación y los sumadores en cascada. Los detalles de implementación para una ventana de 3×3 y 5×5 con configuración simétrica y no simétrica sobre un FPGA *Spartan-6 LX45* se muestran en la Tabla 1, mientras, que los resultados de la normalización de las operaciones matemáticas se muestran en la Tabla 2.

Tabla 1

Consumo de recursos del bloque *XIL Convolution*: análisis de simetría.

Recursos (*Spartan-6 LX45*)		**XIL Convolution**			
		Average 3×3		**Average 5×5**	
Tipo	**Disponibles**	**Simétrica**	**No simétrica**	**Simétrica**	**No simétrica**
Slices	**6 822**	45	48	161	136
Registros	**54 576**	48	48	160	160
LUT	**27 288**	136	183	350	477
DSP48A	**58**	6	9	15	25
Frecuencia (*MHz*)		333,000	333,000	280,899	333,000

La configuración simétrica, en comparación con la no simétrica, reduce en $p(q-1)/2$ los bloques de multiplicación mientras que el consumo de recursos lógicos muestra una reducción de un 6,25% de los slices para la configuración simétrica de 3×3 y un incremento de un 18,28% para la configuración simétrica de 5×5. Este incremento se debe a la sustitución de los operadores de multiplicación por sumas en la configuración simétrica. En cuanto a la frecuencia de trabajo se logra alcanzar el procesado de un píxel cada 3,57 ηs para el peor caso.

El análisis de la opción de normalización de las operaciones depende del núcleo de convolución empleado. Para el caso de la operación promedio, el núcleo no normalizado, mostrado en la Figura 8a, presenta todos los valores en uno por lo que se aplican las opciones para reducir los multiplicadores, operación imposible cuando se normaliza el núcleo (Figura 8b). La normalización de las operaciones del bloque de convolución genérica presenta un consumo de recursos superior en 18 y 43

slices para ventanas de 3×3 y 5×5 respectivamente en comparación con la versión no normalizada. Además se emplean nueve y 25 bloques de multiplicación en la configuración no simétrica normalizada mientras que los mismos son eliminados en la configuración no normalizada, incrementando, esta última, la frecuencia de procesado en 42 *MHz* .

Tabla 2

Consumo de recursos del bloque *XIL Convolution*: análisis de normalización.

Recursos (*Spartan-6 LX45*)		**XIL Convolution**			
		Average 3×3		**Average 5×5**	
Tipo	**Disponibles**	**Normalizado**	**No normaliz.**	**Normalizado**	**No normaliz.**
Slices	**6 822**	48	30	136	93
Registros	**54 576**	48	48	160	160
LUT	**27 288**	183	103	477	318
DSP48A	**58**	9	0	25	0
Frecuencia (*MHz*)		333,000	375,093	333,000	375,093

4.-CONCLUSIONES

Se ha descrito un procedimiento que permite la modificación automática de la arquitectura hardware de un bloque de procesado desarrollado para el flujo de diseño de MATLAB®/Simulink®/Xilinx System Generator. Este proceso se realiza mediante un código en MATLAB® para la instanciación, eliminación, interconexión y configuración de los módulos internos que componen un bloque de procesado, modificando así su estructura hardware y el funcionamiento de bloque. La inclusión de estos segmentos de código como parte de la biblioteca *XIL XSGImgLib* simplifica el uso de los bloques de procesado y libera al diseñador de conocer los detalles específicos de sus arquitecturas. Las opciones de implementación hardware disponibles para el bloque de convolución genérico de la biblioteca *XIL XSGImgLib* hacen posible la reconfiguración automática de su arquitectura mediante la ejecución del procedimiento descrito y el análisis de la estructura del núcleo de convolución, permitiendo ajustar el consumo de recursos de la implementación. El empleo de un núcleo simétrico reduce $p(q-1)/2$ bloques de multiplicación en comparación con la versión no simétrica, mientras que la presencia de valores reducibles en el núcleo conlleva a la sustitución o eliminación de bloques de multiplicación que, adicionalmente permiten incrementar la frecuencia de procesado del bloque.

El procedimiento de reconfiguración automática de la arquitectura se aplica a muchos otros bloques de la biblioteca *XIL XSGImgLib*, pudiendo ser extendido a otras bibliotecas de procesado.

REFERENCIAS

1. Bradski G, Kaehler A. Learning OpenCV. 1st ed. Loukides M, Monaghan R, editors. Gravenstein Highway North, Sebastopol, CA: O'Reilly Media, Inc.; 2008. 571 p.
2. González RC, Woods RE, Eddins SL. Digital Image Processing using MATLAB. 2nd ed. USA: Gatesmark Publishing; 2009. 827 p.
3. Grimm F, Bunke H, Hählen J. An approach to expert systems for image processing software libraries. Mathematics and Computers in Simulation. 1994;36:303–13.
4. Pulli K, Baksheev A, Kornyakov K, Eruhimov V. Real-time computer vision with OpenCV. Communications of the ACM. 2012;55(6):61–9.
5. Toledo Moreo A, Navarro Lorente P, Soto Valles F, Suardíaz Muro J, Fernández Andrés C. Experiences on developing computer vision hardware algorithms using Xilinx System Generator. Microprocessors and Microsystems. 2005;29:411–9.
6. Bailey DG. Design for Embedded Image Processing on FPGAs. 1st ed. Solaris South Tower, Singapore: John Wiley & Sons (Asia) Pte Ltd; 2011. 496 p.
7. Hiraiwa J, Amano H. An FPGA Implementation of Reconfigurable Real-Time Vision Architecture. 27th International Conference on Advanced Information Networking and Applications Workshops. Barcelona, Spain: IEEE; 2013. p. 150–5.

8. Gorgon M, Tadeusiewicz R. Hardware-based image processing library for Virtex FPGA. Reconfigurable Technology: FPGAs for Computing and Applications II. 2000;4212:1–10.
9. Genovese M, Napoli E. An FPGA-based Real-time Background Identification Circuit for 1080p Video. Eighth International Conference on Signal Image Technology and Internet Based Systems. Naples: IEEE; 2012. p. 330–5.
10. Wiatr K, Jamro E. Implementation image data convolutions operations in FPGA reconfigurable structures for real-time vision systems. International Conference on Information Technology: Coding and Computing. Las Vegas, NV, USA: IEEE Comput. Soc; 2000. p. 152–7.
11. Toledo Moreo A, Cuenca-Asensi S, Suardíaz Muro J. Codesign Environment for Computer Vision Hw/Sw Systems. In: Cristóbal G, Javidi B, Vallmitjana S, editors. 5th International Workshop on Informational Optics. Toledo, Spain: AIP; 2006. p. 527–36.
12. Toledo Moreo A, Suardíaz Muro J, Cuenca-Asensi S. Entorno de codiseño para sistemas heterogéneos de procesamiento de imagen. XXVI Jornadas de Automática. Alicante - Elche, España: Comité Español de Automática CEA-IFAC; 2005. p. 1113–20.
13. Vicente-Chicote C, Toledo Moreo A, Sánchez-Palma P. Image Processing Application Development: From Rapid Prototyping to SW/HW Co-simulation and Automated Code Generation. In: Marques JS, Pérez de la Blanca N, Pina P, editors. Second Iberian Conference, Pattern Recognition and Image Analysis Lecture Notes in Computer Science. Estoril, Portugal: Springer Berlin Heidelberg; 2005. p. 659–66.
14. Garcés-Socarrás LM, Sánchez-Solano S, Brox Jiménez P, Cabrera Sarmiento AJ. Library for model-based design of image processing algorithms on FPGAs. Revista de la Facultad de Ingeniería Universidad Antioquia. 2013;1(68):36–47.
15. Toledo Moreo A, Vicente-Chicote C, Suardíaz Muro J, Cuenca-Asensi S. Xilinx System Generator Based HW Components for Rapid Prototyping of Computer Vision SW/HW Systems. In: Marques JS, Pérez de la Blanca N, Pina P, editors. Second Iberian Conference, Pattern Recognition and Image Analysis Lecture Notes in Computer Scienceond Iberian Conference. Estoril, Portugal: Springer Berlin Heidelberg; 2005. p. 667–74.
16. Toledo Moreo A, Suardíaz Muro J, Cuenca-Asensi S, Grediaga A. Novel Simulink Blockset for Image Processing Codesign. IEEE Mediterranean Electrotechnical Conference. Málaga, Spain: IEEE; 2006. p. 117–20.
17. Couprie M. PINK image processing library. Meeting on Image Processing Libraries. Paris, France; 2012. p. 1–4.
18. Tschumperlé D. The CImg Library. Meeting on Image Processing Libraries. Paris, France; 2012. p. 1–4.
19. Maddocks J, Williams R. VHDL Image Processing Source Modules [Internet]. Hunt Engineering. Brent Knoll, Somerset, UK: Hunt Engineering; 2006. Available from: http://www.hunteng.co.uk/pdfs/tutor/FPGAImagingReference.pdf
20. Xilinx. Xilinx CORE Generator System [Internet]. www.xilinx.com. 2014 [cited 2015 Jan 21]. p. 1. Available from: http://www.xilinx.com/tools/coregen.htm
21. Altera. Video and Image Processing Suite. San José, CA, USA: Altera Corporation; 2014. 239 p.
22. Eyetronic N. PIXTERA Hardware-Accelerated Image Co-Processor Reference Design. Marseille, France; 2011.
23. Xilinx. System Generator for DSP Referece Guide. San José, CA, USA: Xilinx Inc.; 2013. 606 p.
24. Xilinx. Vivado Design Suite User Guide. San José, CA, USA: Xilinx Inc.; 2014. 660 p.
25. Garcés-Socarrás LM, Brox Jiménez P, Sánchez-Solano S, Cabrera Sarmiento AJ. Librería de módulos IP para la implementación sobre FPGA de algoritmos de procesado de imágenes. XI Jornadas de Computación Reconfigurable y Aplicaciones. Universidad de La Laguna, Tenerife, España; 2011. p. 227–34.
26. MathWorks. Create Dynamic Mask Dialog Boxes [Internet]. MATLAB® & Simulink® - Documentation. 2014 [cited 2014 Nov 18]. p. 1. Available from: http://www.mathworks.com/help/simulink/ug/create-dynamic-mask-dialog-boxes.html
27. MathWorks. Control Masks Programmatically [Internet]. MATLAB® & Simulink® - Documentation. 2014 [cited 2014 Nov 18]. p. 1. Available from: http://www.mathworks.com/help/simulink/ug/control-masks-programmatically.html
28. MathWorks. Create Dynamic Masked Subsystems [Internet]. MATLAB® & Simulink® - Documentation. 2014 [cited 2014 Nov 18]. p. 1. Available from: http://www.mathworks.com/help/simulink/ug/create-dynamic-

masked-subsystems.html

29. Popinchalk S. Advanced Masking Concepts [Internet]. MATLAB® Central - Guy and Seth on Simulink®. 2008 [cited 2014 Nov 18]. p. 1. Available from: http://blogs.mathworks.com/seth/2008/08/05/advanced-masking-concepts/
30. Popinchalk S. How To Make Your Own Simulink Block [Internet]. MATLAB® Central - Guy and Seth on Simulink®. 2008 [cited 2014 Nov 18]. p. 1. Available from: http://blogs.mathworks.com/seth/2008/07/27/how-to-make-your-own-simulink-block/
31. Popinchalk S. Dynamic Mask Dialogs [Internet]. MATLAB® Central - Guy and Seth on Simulink®. 2008 [cited 2014 Nov 18]. p. 1. Available from: http://blogs.mathworks.com/seth/2008/08/13/dynamic-mask-dialogs/
32. Popinchalk S. Mask Initialization and Self-Modifying Blocks [Internet]. MATLAB® Central - Guy and Seth on Simulink®. 2008 [cited 2014 Nov 18]. p. 1. Available from: http://blogs.mathworks.com/seth/2008/08/21/mask-initialization-and-self-modifying-blocks/
33. Popinchalk S. Libraries in Simulink [Internet]. MATLAB® Central - Guy and Seth on Simulink®. 2008 [cited 2014 Nov 18]. p. 1. Available from: http://blogs.mathworks.com/seth
34. González RC, Woods RE. Digital Image Processing. 3rd ed. Horton MJ, McDonald M, Dworkin A, Opaluch W, Disanno S, Kernan R, editors. Upper Saddle River, New Jersey, USA: Prentice Hall; 2007. 976 p.
35. Turney R. Two-Dimensional Linear Filtering. Xilinx Application Notes. 2007;933(1.1):1–8.

AGRADECIMIENTOS

Esta investigación ha sido financiada parcialmente por la Agencia Española de Cooperación Internacional para el Desarrollo (AECID) mediante los proyectos PCI D/024124/09, PCI D/030769/10 y PCI A1/039607/11 (http://www.imse-cnm.csic.es/fortin), así como por el programa CAPES/MES mediante el proyecto 219/2013.

AUTORES

Luis Manuel Garcés Socarrás, Ingeniero en Automática en el 2006, Máster en Sistemas Digitales en el 2011 por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Auxiliar 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 su tema de investigación doctoral sobre aceleración de algoritmos mediante hardware reconfigurable para el procesado de imágenes y vídeos.

Dirección: Calle 48 Nº 2505 e/ 25 y 27 San Antonio de los Baños. Artemisa. Cuba. CP 32500

Email: lmgarcess@electrica.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 Principal 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.

Email: alex@electrica.cujae.edu.cu

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

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 adscrita al Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica (IMSE-CNM-CSIC), Sevilla, España.

Email: brox@imse-cnm.csic.es

Egídio Ieno Junior, Graduado de Ingeniería Eléctrica por el Instituto Nacional de Telecomunicaciones (INATEL), Santa Rita do Sapucaí, Minas Gerais, Brasil, en 2001. Máster en Telecomunicaciones en 2003 por el mismo instituto. Actualmente es profesor del Centro Federal de Educación Tecnológica de Minas Gerais (CEFET MG) y participante de un proyecto de investigación entre Brasil y Cuba, desarrollando algoritmos de procesado de imágenes basados en FPGA.

Email: egidio_ieno@yahoo.com.br

Tales Cleber Pimenta, Máster en Ingeniería Eléctrica por la Universidad Federal de Itajubá (UNIFEI), Itajubá, Minas Gerais, Brasil. Doctor en Ingeniería Eléctrica por la Universidad de Ohio, 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 para aplicaciones.

Email: tales@unifei.edu.br