DOI: http://dx.doi.org/10.15446/ing.investig.v37n2.62328

Model-based implementation of self-configurable intellectual property modules for image histogram calculation in FPGAs

Implementación de módulos de Propiedad Intelectual modificables para el cálculo de histogramas en FPGA sobre un flujo de diseño basado en modelos

L.M. Garcés-Socarrás[1], D.A. Romero[2], A.J. Cabrera[3], S. Sánchez-Solano[4], and P. Brox[5]

ABSTRACT

This work presents the development of self-modifiable Intellectual Property (IP) modules for histogram calculation using the model-based design technique provided by Xilinx System Generator. In this work, an analysis and a comparison among histogram calculation architectures are presented, selecting the best solution for the design flow used. Also, the paper emphasizes the use of generic architectures capable of been adjustable by a self-configurable procedure to ensure a processing flow adequate to the application requirements. In addition, the implementation of a configurable IP module for histogram calculation using a model-based design flow is described and some implementation results are shown over a Xilinx FPGA *Spartan-6 LX45*.

Keywords: Digital image processing, histogram calculation, FPGA, xilinx system generator, MATLAB®/Simulink®, self-configuration.

RESUMEN

Este artículo presenta el desarrollo de módulos de propiedad intelectual modificables automáticamente para el cálculo de histogramas empleando el flujo de diseño basado en modelos provisto por Xilinx System Generator. En este artículo se realiza un análisis y comparación entre las arquitecturas para el cálculo de histogramas, seleccionando la mejor solución para el flujo de diseño empleado. También se hace énfasis en el uso de arquitecturas genéricas capaces de ajustarse a las necesidades del flujo de datos de la aplicación mediante un procedimiento de configuración automática. Además, se describe la implementación de un módulo de propiedad intelectual configurable para el cálculo de histogramas sobre el flujo de diseño basado en modelos, del cual se muestran algunos detalles de implementación para diferentes opciones de configuración sobre un FPGA *Spartan-6 LX45* de Xilinx.

Palabras clave: Procesado Digital de imágenes, cálculo de histogramas, FPGA, sistema de generación Xilinx MATLAB®/Simulink®, configuración automática.

Received: January 31st 2017
Accepted: July 4th 2017

Introduction

Digital Image Processing (DIP) tasks have, as their main objective, the application of certain mathematical operations over an image to obtain a desired result (González & Woods, 2007). To achieve this objective, many software-based (SW) solutions have been developed in recent decades, using sequential algorithms for General Purpose Processors (GPPs) (Bailey, 2011; Pulli, Baksheev, Kornyakov, & Eruhimov, 2012). On the other hand, hardware-based (HW) implementations - like Field Programmable Gates Arrays (FPGAs) – have been used to increase the operation frequency of DIP systems (Alsuwailem & Alshebeili, 2005; Bailey, 2011; Barranco,

[1] B.Sc. in Automation Engineering. M.Sc. in Digital Systems and D. Sc. in Technical Sciences, Technologic University of Havana "José Antonio Echeverría" (CUJAE), Havana, Cuba. Affiliation: Automation and Computation Department, Technologic University of Havana "José Antonio Echeverría" (CUJAE), Cuba. E-mail: lmgarcess@automatica.cujae.edu.cu

[2] B.Sc. in Automation Engineering, Technologic University of Havana "José Antonio Echeverría" (CUJAE), Havana, Cuba. Affiliation: Instituto Central de Investigaciones Digitales (ICID), Cuba. E-mail: dalejandro@icid.cu

[3] B. Sc. and M.Sc. in Electrical Engineering and D. Sc. in Technical Sciences, Technologic University of Havana "José Antonio Echeverría" (CUJAE), Havana, Cuba. Affiliation: Automation and Computation Department, Technologic University of Havana "José Antonio Echeverría" (CUJAE), Cuba. E-mail: alex@automatica.cujae.edu.cu

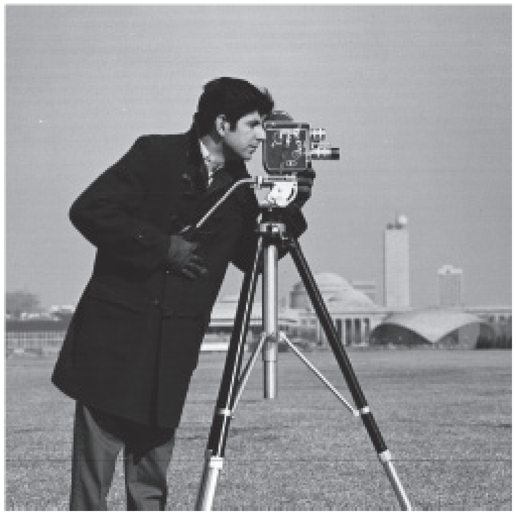
[4] D. Sc in Physis, Universidad de Sevilla, Spain. Affiliation: Instituto de Microelectrónica de Sevilla (CSIC/Universidad de Sevilla), Sevilla, Spain. E-mail: santiago@imse-cnm.csic.es

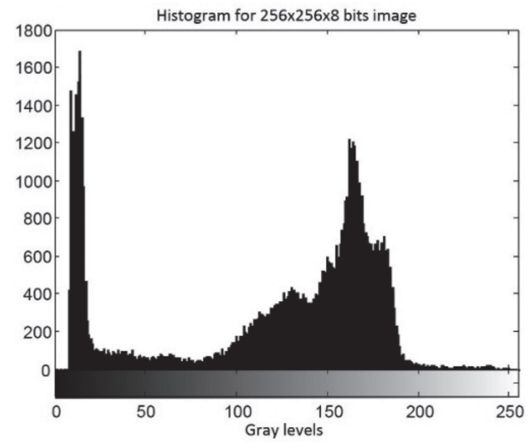
[5] D. Sc in Physis, Universidad de Sevilla, Spain. Affiliation: Instituto de Microelectrónica de Sevilla (CSIC/Universidad de Sevilla), Sevilla, Spain. E-mail: brox@imse-cnm.csic.es

How to cite: Garcés-Socarrás, L.M., Romero, D.A., Cabrera, A.J., Sánchez-Solano, S., Brox, P. (2017). Model-based implementation of self-configurable Intellectual Property modules for image histogram calculation in FPGAs. *Ingeniería e Investigación*, 37(2), 74-81.
DOI: 10.15446/ing.investig.v37n2.62328

Díaz, Gibaldi, Sabatini, & Ros, 2012; Hanumantharaju, Ravishankar, Rameshbabu, & Ramachandran, 2011). These devices are suitable for parallel computing systems that allow the implementation of elaborate functions.

Increasing computing power required by current DIP algorithms can be achieved by performing intensive computing tasks in HW, as well as by exploiting the parallelism of the devices and the partial independence of the algorithms (Bailey, 2011; Qasim, Abbasi, & Almashary, 2009).


a.



b.

Figure 1. Histogram calculation. a) Image under test (256 × 256 × 8-bits). b) Histogram graph.
Source: MATLAB® Software.

The integration of Electronic Design Automation (EDA) tools (Sangiovanni-Vincentelli, 2005) and model-based development frameworks such as MATLAB®/Simulink® has motivated the development of new design techniques for autonomous and modular processing systems, reducing the global design time. System Generator tool (XSG), developed by Xilinx, uses a schematic description to create and parameterize the components of the design, but makes difficult the modification of the internal architecture of the block. MATLAB®/Simulink® delivers some instructions for adding, deleting and interconnecting Simulink® blocks using a complex MATLAB® script, changing the architecture of the block and increasing its versatility (Popinchalk, 2008a, 2008b, 2008c). Those instructions are used to develop a new methodology to create self-configurable Image Processing blocks for XSG (Garcés-Socarrás *et al*., 2016).

Histogram processing is a frequent operation in DIP, showing the statistical distribution of gray or color levels of an image (Bailey, 2011; González & Woods, 2007). The correct manipulation of the histogram permits the equalization of the levels, to obtain a better image. This information is also used for segmentation and image compression (Blair, Robertson, & Hume, 2013; Cho, Jin, Pham, Kim, & Jeon, 2007; González & Woods, 2007; Gu, Noman, Aoyama, Takaki, & Ishii, 2013; Kelly, Siddiqui, Bardak, & Woods, 2014; Kokufuta & Maruyama, 2010; Ma, Najjar, & Roy-Chowdhury, 2014).

Applications of this technique are mostly software-based sequential solutions with limited parallelism, depending on the processing capabilities of the processor and the characteristics of the code executed on it (González & Woods, 2007). Hardware solutions need some modifications to improve resource utilization and exploit HW parallelism. Some histogram calculation architectures for FPGAs use different variations described in the bibliography, which allow choosing an adequate tradeoff between resources consumption and operation frequency (Bailey, 2011; Jamro, Wielgosz, & Wiatr, 2007; Muller, 1995; Shahbahrami, Hur, Juurlink, & Wong, 2008).

The present article describes the implementation of a self-configurable IP module for histogram calculation that could change its internal architecture using a System Generator model-based design flow. This module is part of the image processing toolbox *XIL XSGImgLib* designed for the development of computer vision systems using XSG (Garcés-Socarrás, Sánchez-Solano, Brox Jiménez, & Cabrera Sarmiento, 2013). First, some theoretical concepts about image histogram techniques are presented, analyzing the architectures of the histogram calculation methods, choosing the most adequate for the selected design flow. Then, the implementation of a modifiable IP module by a self-configurable procedure is performed, comparing and evaluating the resource consumption and operating frequency for different configurations. Finally, main conclusions of this work are presented, exposing the advantages of the implementations of highly configurable modules, adaptable to different applications.

Images histogram

A histogram represents a variable in a bar graph, where the height of each bar is proportional to the number of times that appears each specific value or group of values. For a digital image, the histogram provides a graphical representation of the tonal distribution. The histogram function ($H(i)$) of a gray scale image is the quantity of pixels (t_i) for each gray level (i) in the image (González & Woods, 2007). Mathematically, the histogram function (l) is defined as the summatory of the number of pixels with the same gray level (l). Each pixel at the coordinates x and y is denoted as $f(x,y)$ and the image size contains $m \times n$ pixels (Bailey, 2011).

$$H(i) = \sum_{x=0,y=0}^{m-1,n-1} \begin{cases} 1, & f(x,y) = i \\ 0, & f(x,y) \neq i \end{cases} \tag{1}$$

The total number of histogram levels (L) can be reduced by grouping consecutive pixel values in the same interval (Alsuwailem & Alshebeili, 2005; Jamro *et al.*, 2007). In this case, the counting of histogram values in (l) is increased when $\lfloor f(x,y) \times G / L \rfloor = i$, being G the new number of histogram levels. Figure 1 shows an 8-bit gray scale image and its respective histogram graph. The abscissa (horizontal axis) of the graph represents the possible gray levels in the image ($L=G=2^8=256$) while the ordinate (vertical axis) denotes the quantification for each gray level.

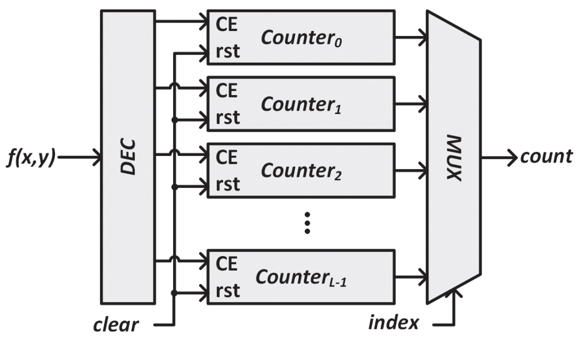
Architectures for histogram calculation

According to its mathematical definition, histogram calculation requires counting the pixels of each color or gray level to access this information in the processing step. Before the quantification process, an initialization stage is needed because all histogram level records should be reset to zero before analyzing a new image. Two different architectures for histogram calculation are applicable for hardware device implementations, where the base for the pixel level accumulation is counter blocks (Figure 2a) or memory blocks (Figure 2b), respectively.

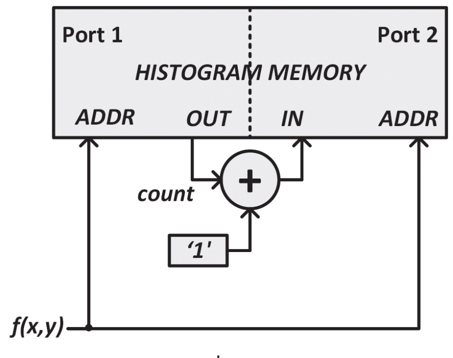
The architecture shown in Figure 2a uses counter blocks to increase calculation speed and to perform the initialization stage on a single clock cycle. With the arrival of a new pixel ($f(x,y)$) at the histogram calculation block, the respective level counter (*Counteri*, where $0 \leq i \leq G-1$) is selected and its value is incremented. Once the image is fully analyzed, values of the *index* signal are appropriately swept so that the *count* output sequentially provides the cumulative values for each color or gray level in the image. This operation requires as many clock cycles as the number of levels considered in the histogram. At the end of the processing cycle, a control system activates the *clear* signal that simultaneously returns all counters to zero, performing the initialization stage of the block.

The main problem in this architecture is the use of many logic resources for the counters array, whose size is equal to the number of color or gray levels selected for the image histogram calculation. Also the counters value width is set, for each counter, to the worst case (when the hole image only has one color or gray level) which is equal to the image size ($m \times n$-pixels). The utilization of the schematic description provided by XSG needs a self-configuration methodology and a generic architecture, explained in (Garcés-Socarrás *et al.*, 2016), for the parameterization of the number of levels for the histogram, meanwhile, the use of HDL techniques is more common in this task.

The dual-port memory solution, as shown in Figure 2b, reduces the logic resources consumption, substituting them by memory blocks, which can be easily parameterized. When a new pixel ($f(x,y)$) from the image arrives at the histogram calculation block, its value is used to address the corresponding color or gray level cell in the memory. Then, the accumulated value of the pixel level is obtained by reading the active memory location using *Port 1*. This value is incremented by one and stored at the same memory location using *Port 2*. Taking into account the hardware perspective, a delayed writing operation at *Port 2* is needed because the cumulative value for this pixel level is read by *Port 1* in the next clock cycle, and it needs to be updated and re-written before a new pixel arrives and another memory location is selected (Bailey, 2011). Once the image is completely analyzed, the control system reads all memory locations for the pixel level accumulation through *Port 1* output, using as many read cycles as color or gray levels were configured for the histogram calculation, at the same time the initialization stage of the memory cells is performed.



a.



b.

Figure 2. Architectures for histogram calculation. a) Using counters; b) Using dual-port memories (Bailey, 2011).
Source: Authors

In this architecture, a simultaneous access to the same memory cell could occur, and then, the value read is not valid in this clock cycle. E. Jamro presented a solution in Jamro *et al*. (2007) where the value of the histogram level for the pixel under test is active during two clock cycles. At the first cycle, the operations of memory reading and the increment of the quantification value are performed and, at the second cycle, the updated quantification value is written in the same memory cell. This solution does not work when the next pixel to analyze has the same color or gray level as the previous pixel. D. Bailey solved this problem in (Bailey, 2011) using a comparator block with two inputs (the previous and the actual pixels); if the previous and the actual pixels are equals, the read operation of the accumulation at *Port 1* is discarded and a new update operation is performed to the previous accumulation value. This solution prevents a read/write operation in the same memory cell when continuous pixels have the same color or gray level, which often occurs in many regions of an image.

Basic histogram calculation architectures use a simple data-flow of pixels where only one histogram counter module is needed (*HistCell1*), as shown in Figure 3a. This architecture works in two steps. In the first step, the whole image is analyzed calculating the histogram, for obtaining the results in the second step. This architecture causes a delay in the processing flow equal to as many clock cycles as pixels in the image, obtaining a new histogram every two images. To solve this issue some authors propose the use of a dual flow architecture (Figure 3b) with two counter modules (Gorgon & Tadeusiewicz, 2000; Maggiani, Salvadori, Petracca, Pagano, & Saletti, 2014). While the first module is active for histogram calculation (*HistCell1*), the second one (*HistCell2*) delivers the result of the histogram of the previous image. Once this process is finished the switches *SW1* and *SW2* commute positions swapping functions of the counter modules, generating a continuous data flow.

The development of a modifiable processing block for simple and dual data-flow requires a self-configurable methodology where the redistribution of Simulink® modules in the architecture is possible (Garcés-Socarrás *et al*., 2016).

Implementation of histogram calculation IP

Xilinx System Generator is a development tool for the design and implementation of embedded systems with basics IP modules and a model-based design technique that speeds up the elaboration of complex systems. The tool performs the process of synthesis and implementation of the design, and the process of configuring the target device automatically from a Simulink® model. This development tool was used to design the IP modules of the image processing toolbox *XIL XSGImgLib*, which provides parameterizable blocks for basic image processing tasks and allows the implementation of advanced DIP technique over FPGAs (Garcés-Socarrás *et al*., 2013). The IP module presented in this article is part of this image processing toolbox.

IP module for histogram calculation

As mentioned previously, the development of IP modules for histogram calculation based on counters use several logic resources. For this reason, the design based on dual-port memories is the solution selected for implementing this block. The modifications to the initial architecture proposed in (Bailey, 2011) are applied to solve read/write accesses to memory cells at the same time. Also, the use of simple or dual data-flow architecture is selected in the configuration of the block, to choose the appropriate data flow for the final application.

Figure 4a shows the generic architecture of the proposed IP module for histogram calculation, which is composed by four main blocks: the histogram controller, two histogram memories (*Hist. DP Memx*) and the output multiplexer (*MUX*). This architecture, saved in a Simulink® file (*.mdl/slx*), is analyzed by a MATLAB® script (*.m*) saving the position and orientation of all modules in the architecture into a MATLAB file (*.mat*) (Figure 5), making a SW description of a graphical architecture (Garcés-Socarrás *et al*., 2016).

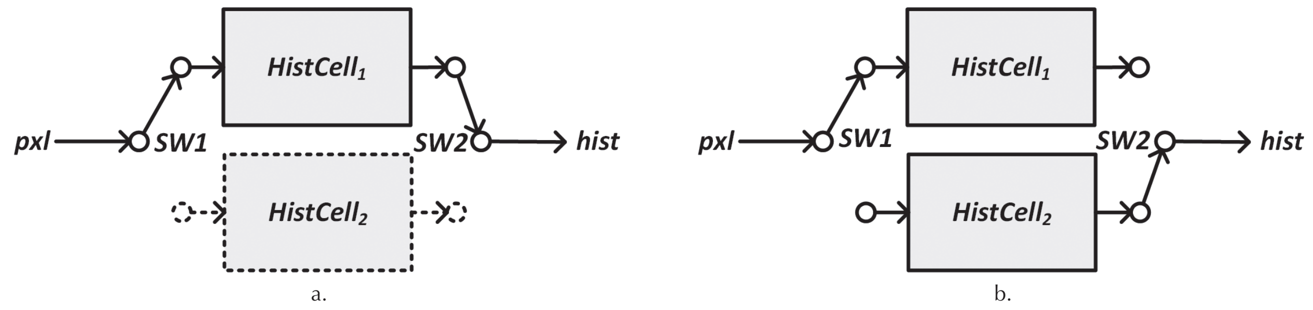
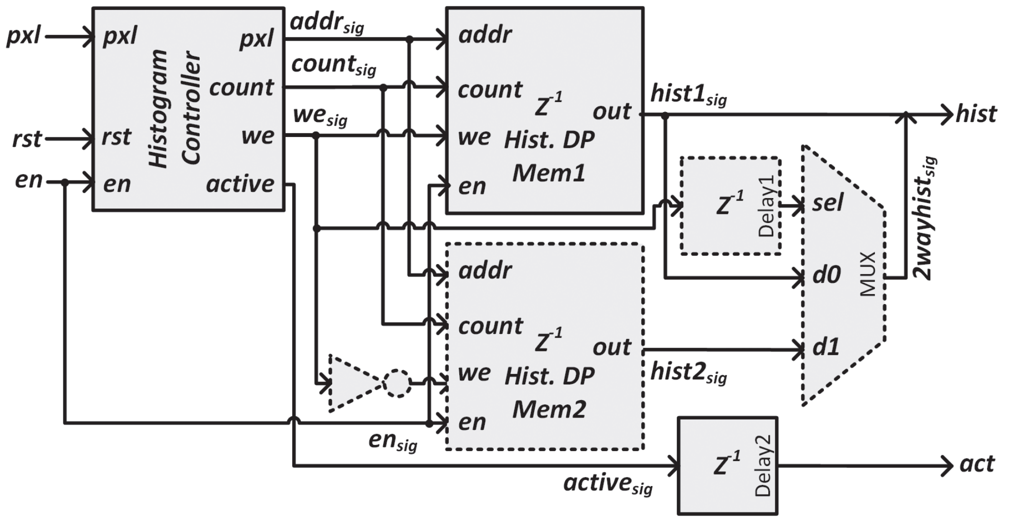
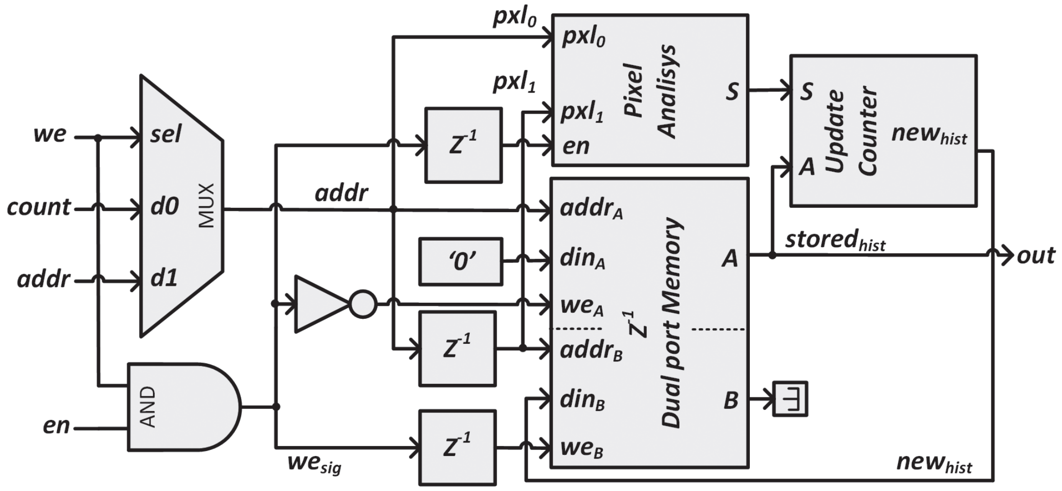


Figure 3. Architectures for histogram calculation. a) Simple data-flow. b) Dual data-flow.
Source: Authors



a.



b.

Figure 4. Architectures of histogram calculation. a) Histogram calculation IP Block. b) Dual-port histogram module.
Source: Authors

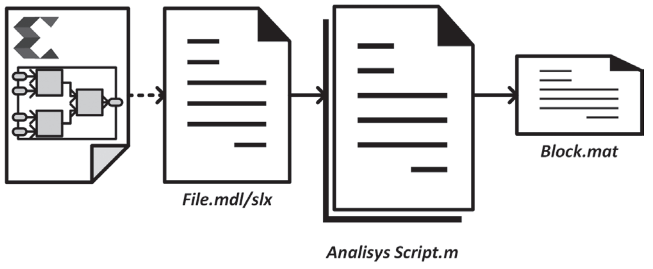


Figure 5. Analysis of a generic architecture.
Source: Authors

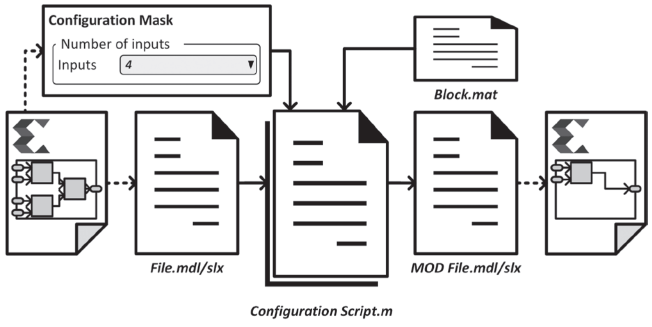


Figure 6. Modification of an architecture.
Source: Authors

When the designer, using the configuration mask of the processing block (Figure 6), selects a different memory architecture, a configuration script analyses the structure of the Simulink® file, detecting which modules have to be erased and which have to be added, and creates a modified Simulink® file (*MOD File.mdl/slx*) using the data previously stored in MATLAB® workspace (*Block.mat*).

In a simple data-flow (*1 – Way*), only one histogram memory module is needed (*Hist. DP Mem$_1$*), deleting the second one (*Hist. DP Mem$_2$*) and the output multiplexer (*MUX*) by the configuration script, while, when a dual data-flow is selected (*2 – Way*) by the designer those modules are added automatically into the architecture reducing the processing delay between images.

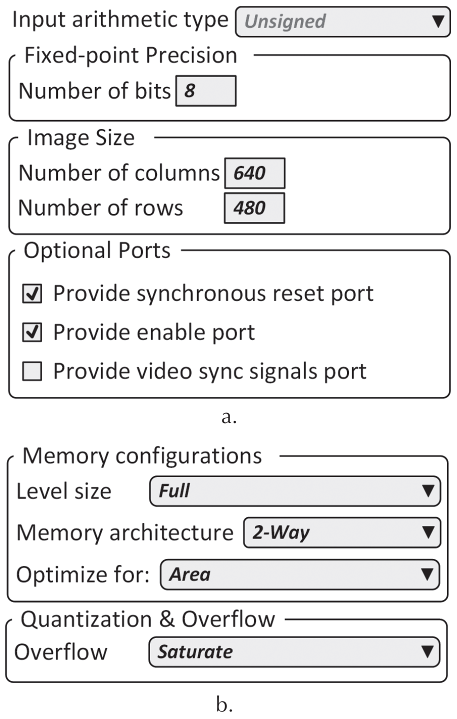
The first block (*Histogram Controller*) receives the input pixel (*pxl*), as well as the enable (*en*) and the initialization (*rst*) signals to handle the flow of image pixels into the module. Once the image is being processed (*en*=*'1'*), the control block classifies the pixels according to the number of histogram levels chosen by the designer. It generates the address signal of each pixel ($addr_{sig}$) for the writing operation, the counter signal ($count_{sig}$) for mapping the histogram levels in the reading operation, and the write enable signal (we_{sig}) to switch the histogram dual-port memory blocks operation and commute the histogram outputs ($histx_{sig}$). The control block also generates the validation signal ($active_{sig}$) to indicate a valid histogram value, which allows to synchronize other modules in the image processing system.

Dual-port histogram memory block (Figure 4b) is composed by a multiplexer (*MUX*), a dual-port memory, a pixel analysis unit and an update counter module. The multiplexer selects the input of the memory from the address signal ($addr_{sig}$) or the counter signal ($count_{sig}$), both coming from the controller module. When the memory is set for write operation (we_{sig}=*'1'*), *port A* of the memory is used to obtain the current value of the histogram for the input pixel ($addr_{sig}$), and *port B* to update this value when the next pixel arrives. The pixel analysis compares the current (pxl_0) and the previous (pxl_1) pixels to ensure the correct delayed writing for continuous pixels values, and produces a control signal (*S*) to the update module. This signal has one clock cycle delay, in order to synchronize the system with the memory access, and it is high when the current pixel is equal to the previous one, indicating that the update module has to reject the value read from the memory and the update module has to increase the value of the previous operation. This block redirects the result (new_{hist}) to *port B* data input (din_B) to refresh the previous value.

When the memory is set for reading operation (we_{sig}=*'0'*), the memory receives the counter signal ($count_{sig}$) to sweep all locations, obtaining the histogram values ($stored_{hist}$) from *port A*. At the same time, the initialization process of the memory cells is performed, writing a zero value to each cell.

The configuration mask of the IP is shown in Figure 7, using basic and advanced parameters. Basic parameters allow to define the input pixel precision, the image size and optional control signals to the module, while advanced parameters configure the memory architecture and the overflow method used when a pixel is greater than the pixel precision defined in the basic parameters. The size of the image under test determines the maximum value that could be stored in the memory cells, which should be configured for the worse case (when the images only have one color or grey level). To reduce the quantity of memory cells used for the histogram calculation, this IP also allows the parameterization of the number of levels considered for the histogram (*Level Size*) being usually power of 2 factor.

Table 1 shows the resource consumption for this IP module with different numbers of histogram levels for simple and dual-flow over a *Spartan-6 LX45* FPGA and a 512×384-pixels image. The first column displays the resource type and in parenthesis the total available of each one in the FPGA. Look-up tables (LUTs) and flip-flops (FFs) consumption are reduced when the levels for histogram quantification are lower and also when the architecture changes from dual data-flow to simple data-flow. The use of memory blocks (BRAM16) is constant for each data-flow, because the maximum amount of memory needed for the gray scale image under test is 26-bits (obtained from the quantity of block cells and the width of each one for a 512×384-pixels image), which does not fulfill one memory block that contains 18-kbits per block (Xilinx, 2010). The operating frequency for the IP module is 165,262 *MHz* for the worst case in the dual data-flow, which allows performing real-time operations over high definition images. Comparing dual data-flow and simple data-flow frequencies, dual-memory architecture provides little increment of the processing speed and also requires two times more memory. The most important goal of dual-flow architecture is the continuous output flow that allows to process video sequences.


Input arithmetic type: *Unsigned*

Fixed-point Precision
Number of bits: *8*

Image Size
Number of columns: *640*
Number of rows: *480*

Optional Ports
☑ Provide synchronous reset port
☑ Provide enable port
☐ Provide video sync signals port

a.

Memory configurations
Level size: *Full*
Memory architecture: *2-Way*
Optimize for: *Area*

Quantization & Overflow
Overflow: *Saturate*

b.

Figure 7. Configuration or the histogram calculation IP. a) Basic parameters; b) Advanced parameters.
Source: Authors

Table 1. Resource consumption for Histogram calculation IP module: analysis of data-flow and histogram levels

Resources	Dual flow			Simple Flow
XC6SLX45	Full	Half	Sixteenth	Full
FFs (54,576)	92	89	80	61
LUTs (27,288)	141	139	135	85
BRAM16 (116)	2	2	2	1
Frequency (MHz)	171,174	171,174	165,262	167,280

Source: Authors

Conclusions

In this article a modifiable histogram calculation IP module using System Generator model-based design flow and a self-configuration procedure are presented, compatible with *XIL XSGImgLib* toolbox (available on https://www.researchgate.net/project/XIL-XSGImgLib-Biblioteca-de-procesado-de-imagenes-y-videos-para-System-Generator) for Xilinx FPGAs. This IP allows to speed up the design process of complex computer vision systems, adjusting the resource consumption for the application requirements.

Counter-based architectures for histogram calculation require several logic resources. On the contrary, memory-based architectures allow the reduction of logic resources, but several modifications are needed to avoid simultaneous access at the same memory cells. Dual-flow architecture is an advantage over simple-flow for video processing because it provides a continuous output flow, even when it uses two times more BRAM blocks, and simple-flow architecture is a better solution for a static image processing. So, a self-modifiable processing block is a versatile improvement to adjust the resource consumption according to the application. The selection of the histogram levels allows the reduction of the quantity of memory cells used for the implementation of the histogram calculation block, with a compromise between the number of levels and the variations obtained in the processing blocks connected to this one, like histogram equalization and threshold calculation. This difference is caused by the grouping of levels in the histogram calculation that are reflected in the next processing step.

Acknowledgment

The authors acknowledge the Spanish Agency for International Development Cooperation (AECID) for partially funding this work in the projects PCI D/024124/09, PCI D/030769/10 and PCI A1/039607/11 (http://www.imse-cnm.csic.es/fortin), as well the CAPES/MES program with the project 219/2013.

References

Alsuwailem, A. M., & Alshebeili, S. A. (2005). *A new approach for real-time histogram equalization using FPGA*. In International Symposium on Intelligent Signal Processing and Communication Systems (pp. 397–400). Hong Kong: IEEE. http://doi.org/10.1109/ISPACS.2005.1595430

Bailey, D. G. (2011). *Design for Embedded Image Processing on FPGAs (1st ed.)*. Solaris South Tower, Singapore: John Wiley & Sons (Asia) Pte Ltd. http://doi.org/10.1002/9780470828519

Barranco, F., Díaz, J., Gibaldi, A., Sabatini, S. P., & Ros, E. (2012). *Vector disparity sensor with vergence control for active vision systems*. Sensors, 12, 1771–1799. http://doi.org/10.3390/s120201771

Blair, C., Robertson, N. M., & Hume, D. (2013). *Characterizing a Heterogeneous System for Person Detection in Video Using Histograms of Oriented Gradients: Power Versus Speed Versus Accuracy*. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 3(2), 236–247.

Cho, J. U., Jin, S. H., Pham, X. D., Kim, D., & Jeon, J. W. (2007). *FPGA-Based Real-Time Visual Tracking System Using Adaptive Color Histograms*. In IEEE International Conference on Robotics and Biomimetics (pp. 172–177). Sanya, China: IEEE. http://doi.org/10.1109/ROBIO.2007.4522155

Garcés-Socarrás, L. M., Cabrera Sarmiento, A. J., Sánchez-Solano, S., Brox Jiménez, P., Ieno, E., & Pimenta, T. C. (2016). *Modificación automática de arquitecturas de módulos hardware de procesado de imágenes*. Revista de Ingeniería Electrónica, Automática Y Comunicaciones, XXXVII(3/2016), 21–33. Retrieved from http://rielac.cujae.edu.cu/index.php/rieac/article/view/406

Garcés-Socarrás, L. M., Sánchez-Solano, S., Brox Jiménez, P., & Cabrera Sarmiento, A. J. (2013). *Library for model-based design of image processing algorithms on FPGAs*. Revista de La Facultad de Ingeniería Universidad Antioquia, 1(68), 36–47.

González, R. C., & Woods, R. E. (2007). *Digital Image Processing*. (M. J. Horton, M. McDonald, A. Dworkin, W. Opaluch, S. Disanno, & R. Kernan, Eds.) (3rd ed.). Upper Saddle River, New Jersey, USA: Prentice Hall.

Gorgon, M., & Tadeusiewicz, R. (2000). *Hardware-based image processing library for Virtex FPGA*. Reconfigurable Technology: FPGAs for Computing and Applications II, 4212, 1–10. http://doi.org/10.1117/12.402510

Gu, Q., Noman, A. Al, Aoyama, T., Takaki, T., & Ishii, I. (2013). *A Fast Color Tracking System with Automatic Exposure Control*. In 7th International Conference on Information and Automation or Sustainability (pp. 1–6). Yinchuan, China: IEEE.

Hanumantharaju, M. C., Ravishankar, M., Rameshbabu, D. R., & Ramachandran, S. (2011). *A novel FPGA implementation of adaptive color image enhancement based on HSV color space*. In 3rd International Conference on Electronics Computer Technology (ICECT 2011), Kanyakumari (Vol. 2, pp. 160–163). Kanyakumari: IEEE. http://doi.org/10.1109/ICECTECH.2011.5941676

Jamro, E., Wielgosz, M., & Wiatr, K. (2007). *FPGA Implementaton of Strongly Parallel Histogram Equalization*. In IEEE Design and Diagnostics of Electronic Circuits and Systems (pp. 1–6). Krakow: IEEE. http://doi.org/10.1109/DDECS.2007.4295260

Kelly, C., Siddiqui, F. M., Bardak, B., & Woods, R. (2014). *Histogram of Oriented Gradients front end processing: an FPGA Based Processor Approach*. In IEEE Workshop on Signal Processing Systems (pp. 1–6). Belfast: IEEE. http://doi.org/10.1109/SiPS.2014.6986093

Kokufuta, K., & Maruyama, T. (2010). *Real-time processing of contrast limited adaptive histogram equalization on FPGA*. In 20th International Conference on Field Programmable Logic and Applications (pp. 155–158). Milano, Italy: IEEE Computer Society. http://doi.org/10.1109/FPL.2010.37

Ma, X., Najjar, W. A., & Roy-Chowdhury, A. K. (2014). *Evaluation and Acceleration of High-Throughput Fixed-Point Object Detection on FPGAs*. IEEE Transactions on Circuits and Systems for Video Technology, 25(6), 1051–1062. http://doi.org/10.1109/TCSVT.2014.2360030

Maggiani, L., Salvadori, C., Petracca, M., Pagano, P., & Saletti, R. (2014). *Reconfigurable architecture for computing histograms in real-time tailored to FPGA-based Smart Camera*. In IEEE 23rd International Symposium on Industrial Electronics (pp. 1042–1046). Istanbul: IEEE. http://doi.org/10.1109/ISIE.2014.6864756

Muller, S. (1995). *A New Programmable VLSI Architecture for Histogram and Statistics Computation in Different Windows*. In International Conference on Image Processing, Washington, DC (pp. 73–76). Washington, DC: IEEE. http://doi.org/10.1109/ICIP.1995.529042

Popinchalk, S. (2008a). *Advanced Masking Concepts*. Retrieved November 18, 2014, from http://blogs.mathworks.com/seth/2008/08/05/advanced-masking-concepts/

Popinchalk, S. (2008b). *Dynamic Mask Dialogs*. Retrieved November 18, 2014, from http://blogs.mathworks.com/seth/2008/08/13/dynamic-mask-dialogs/

Popinchalk, S. (2008c). *Mask Initialization and Self-Modifying Blocks*. Retrieved November 18, 2014, from http://blogs.mathworks.com/seth/2008/08/21/mask-initialization-and-self-modifying-blocks/

Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). *Real-time computer vision with OpenCV*. Communications of the ACM, 55(6), 61–69. http://doi.org/10.1145/2184319.2184337

Qasim, S. M., Abbasi, S. A., & Almashary, B. A. (2009). *An overview of advanced FPGA architectures for optimized hardware realization of computation intensive algorithms*. In International Multimedia, Signal Processing and Communication Technologies (pp. 300–303). Aligarh: IEEE. http://doi.org/10.1109/MSPCT.2009.5164235

Sangiovanni-Vincentelli, A. (2005). *The tides of EDA*. Design & Test of Computers, IEEE, 20(6), 59–75. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1246165

Shahbahrami, A., Hur, J. Y., Juurlink, B., & Wong, S. (2008). *FPGA implementation of parallel histogram computation*. In 2nd HiPEAC Workshop on Reconfigurable Computing (pp. 63–72). Göteborg, Sweden.

Xilinx. (2010). *Spartan-6 FPGA Block RAM*. Datasheet: User Guide. Datasheet, Xilinx Inc.