Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2011, Article ID 712494, 10 pages
doi:10.1155/2011/712494

Research Article

A Genetic Programming Approach to Reconfigure a Morphological Image Processing Architecture

Emerson Carlos Pedrino,[1] José Hiroki Saito,[1] and Valentin Obac Roda[2]

[1] *Computer Science Department, Federal University of São Carlos, Rodovia Washington Luís, km 235, 13565-905 São Carlos, SP, Brazil*
[2] *Department of Electrical Engineering, University of São Paulo, Avenida Trabalhador São Carlense, 400 13566-590 São Carlos SP, Brazil*

Correspondence should be addressed to Emerson Carlos Pedrino, emerson@dc.ufscar.br

Received 20 May 2010; Accepted 10 September 2010

Academic Editor: Elías Todorovich

Mathematical morphology supplies powerful tools for low-level image analysis. Many applications in computer vision require dedicated *hardware* for real-time execution. The design of morphological operators for a given application is not a trivial one. Genetic programming is a branch of evolutionary computing, and it is consolidating as a promising method for applications of digital image processing. The main objective of genetic programming is to discover how computers can learn to solve problems without being programmed for that. In this paper, the development of an original reconfigurable architecture using logical, arithmetic, and morphological instructions generated automatically by a genetic programming approach is presented. The developed architecture is based on FPGAs and has among the possible applications, automatic image filtering, pattern recognition and emulation of unknown filter. Binary, gray, and color image practical applications using the developed architecture are presented and the results are compared with similar techniques found in the literature.

1. Introduction

Morphological image processing is a nonlinear branch in image processing developed by Matheron and Serra in the 1960s, based on geometry and on the mathematical theory of order [1–6]. Morphological image processing has proved to be a powerful tool for binary and grayscale image computer vision processing tasks, such as edge detection, noise suppression, skeletonization, segmentation, pattern recognition, and enhancement [7]. Initial applications of morphological processing were biomedical and geological image analysis problems [8]. In the 1980s, extensions of classical mathematical morphology and connections to other fields were developed by several research groups worldwide along various directions, including computer vision problems, multiscale image processing, statistical analysis, and optimal design of morphological filters, to name just a few.

The basic operations in mathematical morphology are the dilation and the erosion, and these operations can be described by logical and arithmetic operators. Dilation and erosion morphological operators can be represented, respectively, by the sum and subtraction of Minkowski sets [9]:

$$A \oplus B = \cup\{B + a \mid a \in A\}, \tag{1}$$

$$A\Theta(-B) = \cap\{A + b \mid b \in B\}. \tag{2}$$

In (1), A is the original binary image, B is the structuring element of the morphological operation, and $B + a$ is the B displacement by a. Therefore, the dilation operation is obtained by the union of all B displacements in relation to the valid A elements. In (2), $-B$ is the 180° rotation of B in relation to its origin. Therefore, the erosion operation corresponds to intersection of the A displacements by the valid points of $-B$. These ideas can be extended to gray-level image processing using maximum and minimum operators, too [9].

As mentioned by Haralick [10], since mathematical morphology operates with shapes, it becomes a natural processing to deal with problems of identification of image objects based on shape. Some other basic computer vision operations such as edge detection, skeletons, and noise elimination can be performed eroding or dilating objects in an algorithmic way.

In color images, pixels are represented by vector values (RGB, e.g.):

$$P(x, y) = [P1(x, y), P2(x, y), P3(x, y)]^T. \tag{3}$$

Mathematical Morphology is based on the application of lattice theory to spatial structures [11]. The definition of morphological operators needs a totally ordered complete lattice structure. A lattice is a partially ordered set in which any two elements have at least an upper bound (supremum) and a greatest lower bound (infimum). The supremum and the infimum are represented by the symbols $\vee$ and $\wedge$, respectively. Thus, a lattice is complete if every subset of the lattice has a single supremum and a single infimum. Color is known to play a significant role in human visual perception. The application of mathematical morphology to color images is difficult due to the vector nature of the color data. The extension of concepts from grayscale morphology to color morphology must first choose an appropriate color ordering, a color space that determines the way in which colors are represented and an infimum and a supremum operator in the selected color space should also be defined. There are several techniques for ordering vectors. The two main approaches are marginal ordering and vector ordering. In the marginal ordering, each component $P1$, $P2$, or $P3$ is ordered independently and the operations are applied to each channel; unfortunately, this procedure has some drawbacks, for example, producing new colors that are not contained in the original image and may be unacceptable in applications that use color for object recognition. The vector ordering method for morphological processing is more advisable. Only one processing over the three dimensional data is performed using this method. There are several ways of establishing the order, for example, ordering by one component, canonical ordering, ordering by distance, and lexicographical order [12].

Once these orders are defined, then the morphological operators are defined in the standard way. The vector erosion of color image f at pixel x by the structuring element B of size n is [2].

$$\text{EnB}(f)(x) = \{\inf[f(z)],\ z \in n(Bx)\}. \tag{4}$$

The corresponding dilation DnB is obtained by replacing the inf by sup

$$\text{DnB}(f)(x) = \{\sup[f(z)],\ z \in n(Bx)\}. \tag{5}$$

An opening is an erosion followed by a dilation, and a closing is a dilation followed by an erosion.

The design of morphological procedures is not a trivial task in practice [13]. Some expert knowledge is necessary to properly select the structuring element and the morphological operators to solve a certain problem [14]. In the literature, there are several approaches using automatic programming to overcome these difficulties [15–22]; however, they present several drawbacks as a limited number of operators, only regular forms of structuring elements and only morphological instructions, to name just a few.

Genetic programming (GP) is the most popular technique for automatic programming nowadays and may provide a better context for the automatic generation of morphological procedures [23]. GP is a branch of evolutionary computation and artificial intelligence [24–26], based on concepts of genetics and Darwin's principle of natural selection to genetically breed and evolve computer programs to solve problems.

Genetic programming is the extension of the genetic algorithms [27] into the space of programs. That is, the objects that constitute the population are not fixed-length character strings that encode possible solutions to a certain problem. They are programs (expressed as parse trees) that are the candidate solutions to the problem [28, 29].

There are few applications of GP for the automatic construction of morphological operators [14, 23] and for color image processing. Thus, we propose a linear genetic programming approach for the automatic construction of morphological, arithmetic, and logical operators, generating a toolbox named *morph_gen* for the Matlab program. The proposed toolbox can be used for the design of nonlinear filters, image segmentation and pattern recognition of binary, grayscale, and color images. The instructions generated by the toolbox are transferred to a 32-stage pipeline architecture developed in this work, which has been implemented on an FPGA. Some examples of applications are presented, and the results are discussed and compared with other approaches.

This paper is organized as follows; a brief review of the basic concepts of morphological operations and genetic programming is presented in Section 1; a detailed description of the developed system is presented Section 2; results and application examples are presented in Section 3; and Section 4 is the conclusions.

2. Developed System

2.1. Training Process. The developed algorithm for automatic construction of morphological operators uses a linear genetic programming approach that is a variant of the GP algorithm that acts on linear genomes [30, 31]. It operates with two images, an input image and an image containing only features of interest which should be extracted from the input image. The genetic procedure looks for operators' sequences in the space of mathematical morphology algorithms that allow extracting the features of interest from the original image. The operators are predefined procedures from a database that work with particular types of structuring elements having different shapes and sizes. It is also possible to include new operators in the database when necessary. The program output is a linear structure containing the best individual of the final population. The output result from one operator is used as input to the subsequent operator and so on, for example, the sequence "ero_q_3->dil_q_3" performs

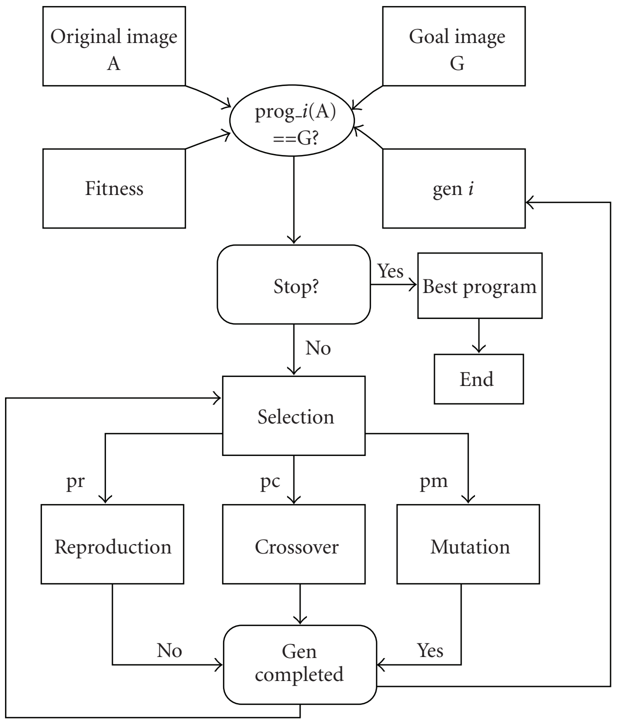

Figure 1: Flowchart of developed system. The index i refers to an individual in the population. The reproduction rate is pr, the crossover rate is pc, and the mutation rate is pm. The goal image can be created using an editor program or a processing program.

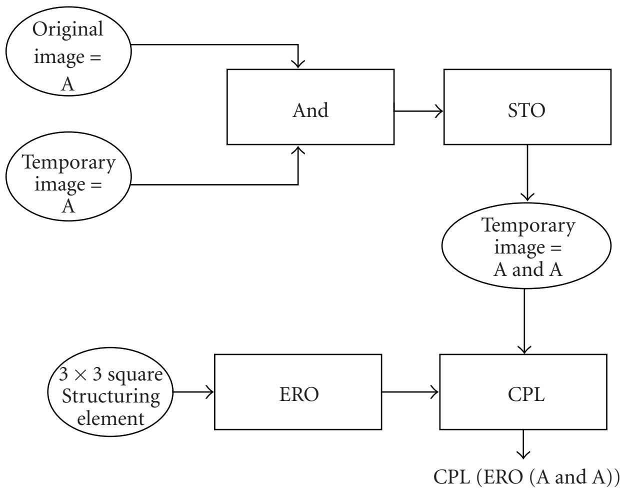

Figure 2: Evaluation of chromosome: "00011011_2" from program: "and-sto-ero-cpl" applied to original image A.

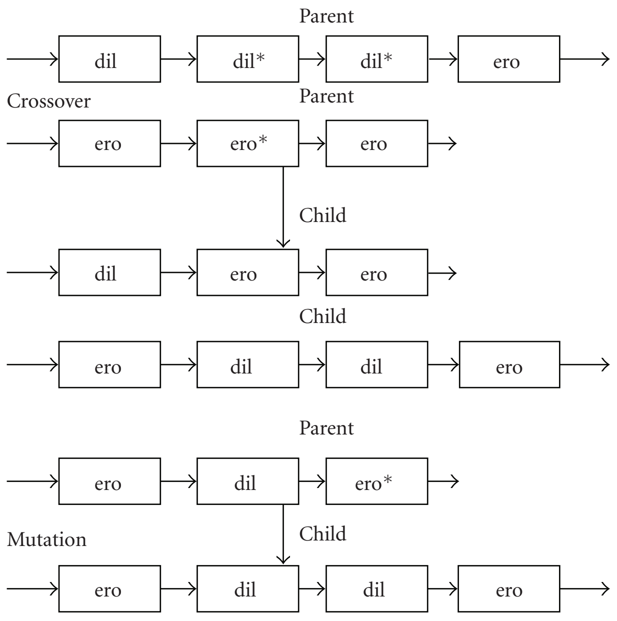

Figure 3: Crossover and mutation operators where $*$ is the selected operator.

an erosion in the input image followed by a dilation using for each operation a 3×3 square structuring element. The genetic algorithm parameters are supplied by the user using a graphical user interface (GUI). The main parameters are: tree depth, number of chromosomes, number of generations, crossover rate, mutation rate, reproduction rate, and certain kinds of operators suited to a particular problem. It has been used for the problems the mean absolute error (MAE) as a fitness measure. The cost function using MAE error was calculated as follows:

$$d(a,b) = \frac{1}{XY}\sum_{i}^{X}\sum_{j}^{Y}\left|a(i,j) - b(i,j)\right|. \qquad (6)$$

In (6), a is the resulting image evaluated by a particular chromosome, b is the goal image with the same size as a, and (i, j) is the pixel coordinate. The chromosomes are encoded as variable binary chains. The main steps of the proposed algorithm are illustrated in Figure 1.

The genetic parameters and the images are supplied by the user; the initial population of programs is randomly generated. Since the chromosomes are encoded as binary chains, if the user has selected the instructions: *and* (AND logic), *sto* (STORE), *ero* (EROSION), and *cpl* (COMPLEMENT), the first operator will be coded as "00_2", the second as "01_2", the third as "10_2", and the last as "11_2". If the chosen tree depth was four, for example, the chromosome: "00011011_2" could be created. The evaluation of this chromosome will be as illustrated in Figure 2, for example, AND $(A,\ A)$ followed by a Store in a temporary variable followed by an Erosion, and followed by a Logical inversion. In example, A is an input binary image. This idea is repeated for the others chromosomes from the initial population.

After evaluation of each chromosome in a generation, a cost value is assigned to each one using (6). The next step is to create a new population of the fittest programs. The selection method used to choose the best individuals for reproduction was the tournament selection [32]. The best ones are selected for the genetic operations of crossover and mutation. In crossover operation, morphological operators are randomly selected and exchanged between parents chromosomes. The mutation operation replaces a randomly selected instruction

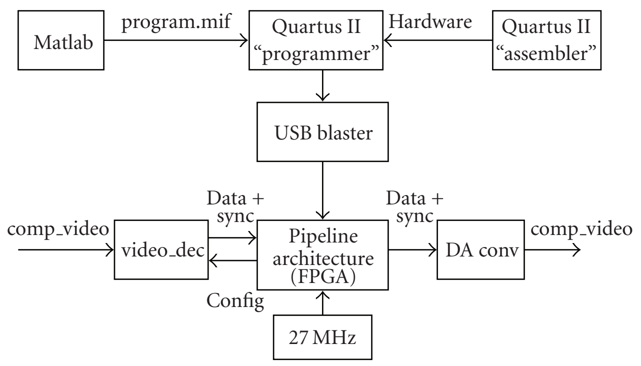

Figure 4: Block diagram of the developed system. After FPGA programming, the composite video is deinterlaced by the video_dec, and the 10-bit RGB data produced are processed by the pipeline architecture. The results are converted again to an analog format to be shown in a VGA monitor.

by another in the range of morphological algorithms space. The reproduction operator copies a single parent into the new generation according to its fitness value. In Figure 3, we can see the crossover and mutation operators used in this paper. This process is repeated for several generations until a stop criteria is reached (the fittest program).

2.2. Implemented Architecture. The block diagram of the developed architecture can be seen in Figure 4. The opcodes (best chromosome) file "program.mif" with the sequence of operators (binary chain) seen in Section 2.1 generated by Matlab is transferred with the other project files containing the description of the architecture to the FPGA board by means of the USB interface from PC through the Quartus II software. In this project, the DE2 board from Altera [33] was used to develop the video architecture that is based on a 32-stage pipeline. The DE2 board contains a Cyclone II (2C35) FPGA, a NTSC/PAL TV decoder circuit, and a VGA output circuit. A composite video signal supplied by a commercial video camera is deinterlaced and converted to 10 bit RGB data (640 × 480 pixels) through a video decoder stage. The RGB frames are processed through the pipeline stages, and the results are converted to an analog format again through a DA converter. Then, the processed images can be shown in a VGA monitor. A 27 MHz oscillator was used as a clock source.

In Figure 5, a block diagram of the pipeline stages is presented. The opcodes from *morph_gen* toolbox are loaded into the stages through a state machine named ROM that contains the original program. The implementation of a stage from the pipeline can be seen in Figure 8, and it is described below.

The state machine ROM uses the bus *dat* and the bus *add* to distribute the data (instructions) to each processor that uses an add (address) in the architecture. For example, the *P*1 has the add = 01 h, the *P*2 has the add = 02 h, and so on. The "program.mif" contains a binary chain representing the chromosome generated by Matlab where each line corresponds to an instruction according to Table 1 that will be processed by each stage. The block diagram of ROM unit explained before can be seen in Figure 6. In Figure 7, a simulation of this unit is shown. Considering Figure 6, after the reset process of the architecture, while the *end_add* pin is low, the state machine loads the instructions referring to a certain problem into the instruction register (IR) of each processor of the pipeline. When the load process ends, the *end_add* pin will be high and the state machine will indicate the timing of the video processing, and this cycle will be repeated when the state machine reads a reset state again.

Each stage stores two adjacent 640-pixel lines followed by a 3-pixel line to constitute a (3 × 3) input to a morphological processor implemented in that stage. This same structure is used by a previously stored result that is delayed by each stage, too. This result is stored in *img_temp* register. Figure 8 shows the block diagram of a stage from the pipeline. Each stage has been built using the Verilog language. The *Instr_dec* block in the processor decodes the instruction stored in IR register and apply a morphological or logical operation, according to Table 1, to input pixels *p1_1, p1_2, p1_3, p2_1, p2_2, p2_3, p3_1, p3_2, and p3_3* (input window) and/or *s2_2* (previous stored result from n-1 stage). In *Instr_dec* block, the dilation and erosion operations are implemented according to (5) and (4), respectively.

For example, the instruction *dil_c_3* (dilation by a 3 × 3 circular structuring element) and *ero_c_3* (erosion by a 3 × 3 circular structuring element) are implemented in Verilog as follows, respectively: *out_dil<=0|p1_2|p2_1|p2_2|p2_3|p3_2*, and out_ero<=1&p1_2&p2_1&p2_2&p2_3&p3_2. To avoid bottlenecks, the system does not use memory access. The only significant delay presented in this architecture is due to the number of the pipeline stages. The logical instructions have been implemented using Verilog HDL through Quartus II. In this architecture, a chromosome is decoded according to Figure 2. Each stage can work with a RGB digital image of 10 bit/channel. For binary processing, the least significant bit of G channel is used. For monochromatic images, the R, G, or B channel is used and for color processing, a combination of R, G, and B, to form a lattice structure required for morphological processing. This combination is as follows: {R1G1B1R2B2G2, . . . , R10G10B10}, thus, RnGnBn is a 30-bit scalar number, and the morphological operations ((4) and (5)) can be defined for color images. After processing, the resulting scalar value is decomposed again into its RGB component.

The implementation idea of the proposed architecture can be seen in the following simplified example (Figure 9) for a dilation of a 5 × 5 binary input image using a 3 × 3 circular structuring element. In this figure, only one stage of the pipeline architecture is shown. Firstly, the image pixels are inserted into the buffers using a raster sweep. The buffers are necessary to maintain a window with the current pixels to be processed in each stage during the raster sweep. Once the structuring element has size 3×3, the first three pixels of each buffer (b1, b2, b3, b6, b7, b8, b11, b12, and b13) are passed to the processor along the time. Since the structuring element, in this hypothetical case, is circular, the only pixels used by the processor are b2, b6, b7, b8, and b12. In this example, a dilation operation that was preconfigured by the state

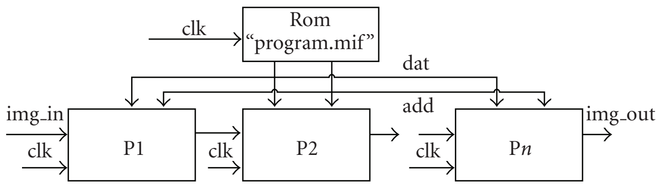

Figure 5: Block diagram of the pipeline stages.

Table 1: Implemented instructions.

Opcode	Name	Comment
0x00	nop	No operation. The input image is copied to the output image.
0x01	dil_q_3	The input image is dilated by a (3 × 3) square structuring element.
0x02	ero_q_3	The input image is eroded by a (3 × 3) square structuring element.
0x03	dil_c_3	The input image is dilated by a (3 × 3) circular structuring element.
0x04	ero_c_3	The input image is eroded by a (3 × 3) circular structuring element.
0x05	dil_h_3	The input image is dilated by a (1 × 3) horizontal structuring element.
0x06	ero_h_3	The input image is eroded by a (1 × 3) horizontal structuring element.
0x07	dil_v_3	The input image is dilated by a (3 × 1) vertical structuring element.
0x08	ero_v_3	The input image is eroded by a (3 × 1) vertical structuring element.
0x09	dil_dd_3	The input image is dilated by a (3 × 3) right diagonal structuring element.
0x0A	ero_dd_3	The input image is eroded by a (3 × 3) right diagonal structuring element.
0x0B	dil_de_3	The input image is dilated by a (3 × 3) left diagonal structuring element.
0x0C	ero_de_3	The input image is eroded by a (3 × 3) left diagonal structuring element.
0x0D	xor1	Exclusive OR between the input image and a temporary image (previous result).
0x0E	Cpl	Logical complement of the input image.
0x0F	sto1	Temporary storage of the input image.
0x10	and1	Logical AND between the input image and a temporary image (previous result).
0x11	or1	Logical OR between the input image and a temporary image (previous result).
0x12	Ldi	Load of the original image.
0x13	cpl_cz	Complement of the input image (grayscale or color).
0x14	add_cz	Arithmetic sum between the input image (grayscale or color) and a temporary image (previous result).

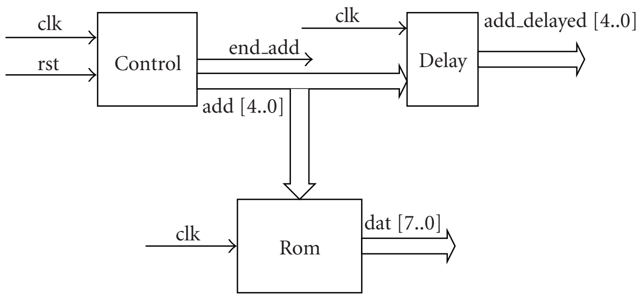

Figure 6: Schematic circuit of ROM unit.

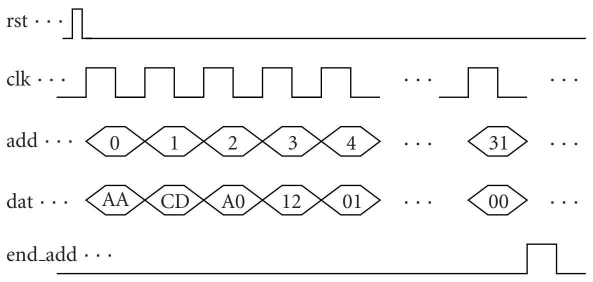

Figure 7: Simulation example of ROM unit.

machine ROM is implemented using a logical OR operator. The output of the stage is given by a stream of pixels. In this specific case, the input active pixels were *img_in (3,2)* and *img_in (3,3)*, thus, after the logical operation, the active pixels of the dilated output image can be seen by means of the result variable.

3. Results and Application Examples

In this section, some results using the developed architecture are presented.

In Figure 10, the input image was corrupted by *salt and pepper* noise with a density of 0.09 generated by Matlab. The instructions *ero_q_3* and *dil_q_3* (morphological operators) were used to construct a morphological filter.

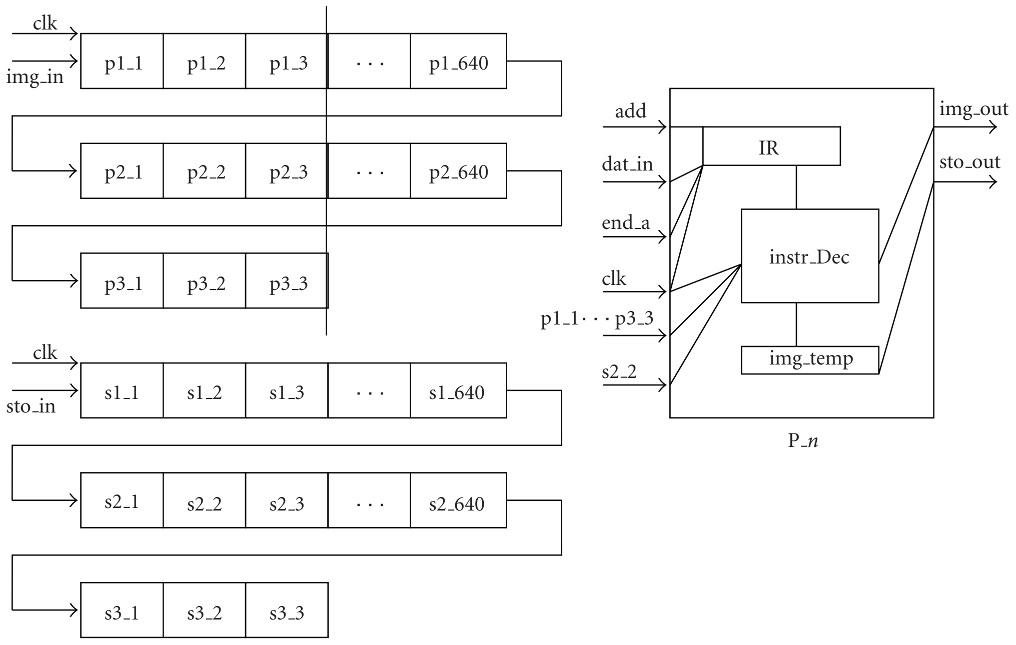

Figure 8: Block diagram of a stage.

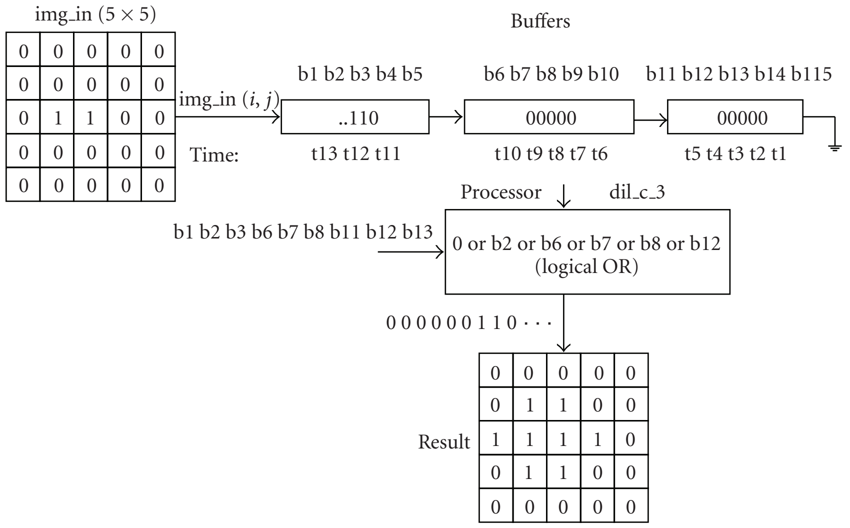

Figure 9: Implementation idea of the proposed architecture.

The genetic procedure converged before the 5th generation and the filter "ero_q_3-> dil_q_3-> dil_q_3-> ero_q_3" (morphological algorithm) was automatically created. The genetic parameters chosen for this task were: 50 generations, 25 chromosomes, depth of tree 4, crossover rate of 90%, mutation rate of 20%, and reproduction rate of 20%. The MAE error found between the goal image and a clear version of the original image was less than 0,4%. The training time was less than 4,3 seconds, and the execution time was performed in real time by the developed system. In this example and in the following ones, a PC notebook equipped with an AMD 64 Athlon processor and 512 MB of system memory was used for the training process.

In Figure 11, an original image and a training image containing features (heads) from a fragment of a music score to be extracted by the evolutionary system are presented. The genetic procedure found the following best program to extract heads using the developed system: "dil_dd_3-> dil_de_3-> dil_dd_3-> dil_v_3-> dil_v_3-> dil_dd_3-> dil_v_3-> ero_q_3-> ero_v_3-> ero_q_3-> ero_c_3". The genetic parameters chosen for this task were: 50 generations, 50 chromosomes, tree of depth 12, crossover rate of 97%, mutation rate of 3%, and reproduction rate of 10%. The MAE error found between the goal image and the obtained result was less than 1,1%. The training time was less than 12 min, and execution time was in real time. This procedure

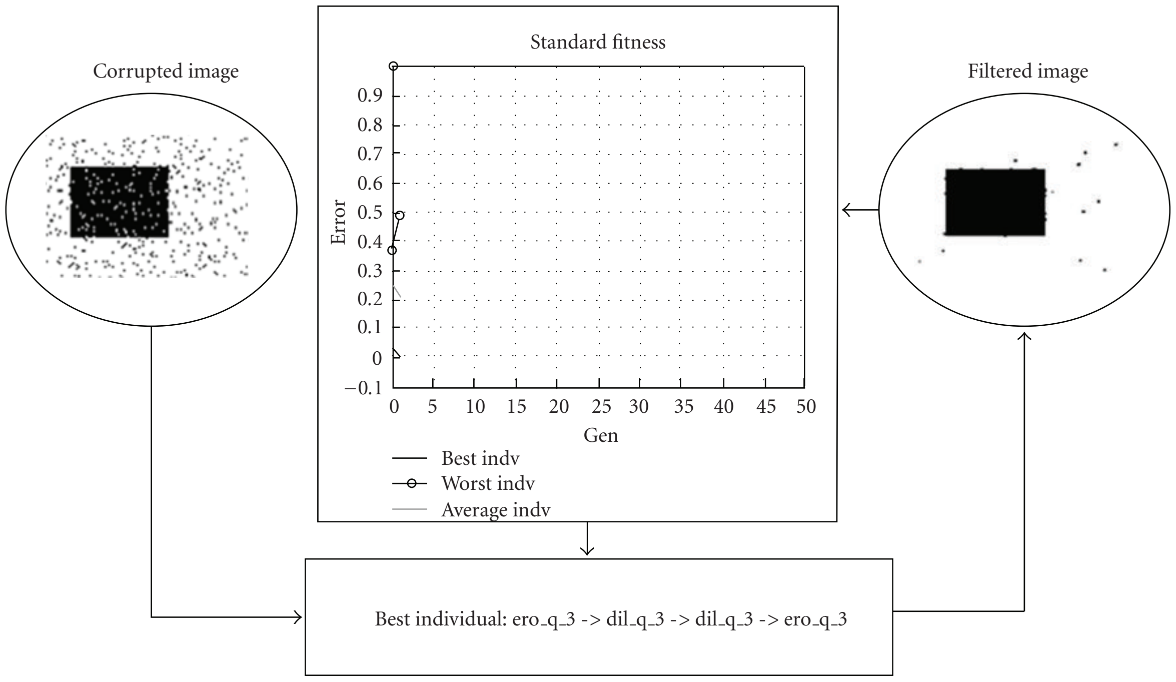

Figure 10: Automatically generated filter to eliminate the salt and pepper noise from the corrupted original image.

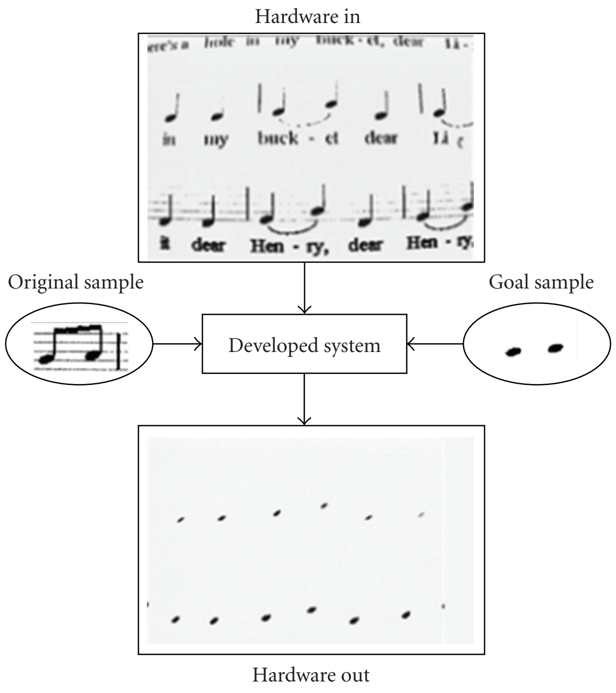

Figure 11: Obtained result by developed hardware for head extraction in real time.

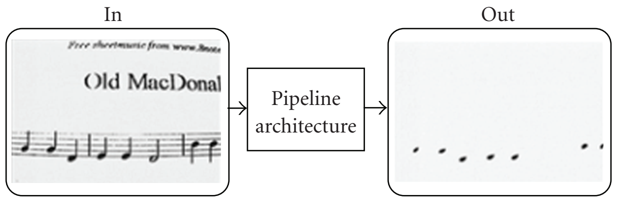

Figure 12: Example of head extraction in real time.

was applied to image in Figure 12 producing an expected result, too.

In Figure 13, an emulation result of the Photoshop's Trace Contour filter after a training process of the evolutionary system is presented. After the presentation of training samples the best program found was "ero_(c_3)-> sto1-> sto1-> dil_(q_3)-> dil_(c_3)-> cpl-> cpl-> dil_c_3-> ero_q_3-> cpl-> or1-> cpl-> cpl." The genetic parameters chosen for this task were 50 generations, 90 chromosomes, tree of depth 16, crossover rate of 97%, mutation rate of 3%, and reproduction rate of 10%. The MAE error found was less than 2,86% compared to Photoshop's result. The training time was less than 18 min, and execution time was in real time.

In Figure 14, there is an example of an emulation result of the Photoshop's Glowing Edge filter generated automatically after a training process of the evolutionary system for the following parameters: 51 generations, 70 chromosomes, tree of depth 9, crossover rate of 95%, mutation rate of 20%, and reproduction rate of 10%. For this task, an intensity image was used as input and the best program found was "add_cz-> add_cz-> dil_c_3-> sto1-> cpl_cz-> dil_c_3-> dil_c_3-> add_cz-> sto1." The MAE error found was less than 6,2% compared to Photoshop's result. In Figure 15 the same result was applied to a color image. The morphological operations in this example preserve the colors in the original image [34].

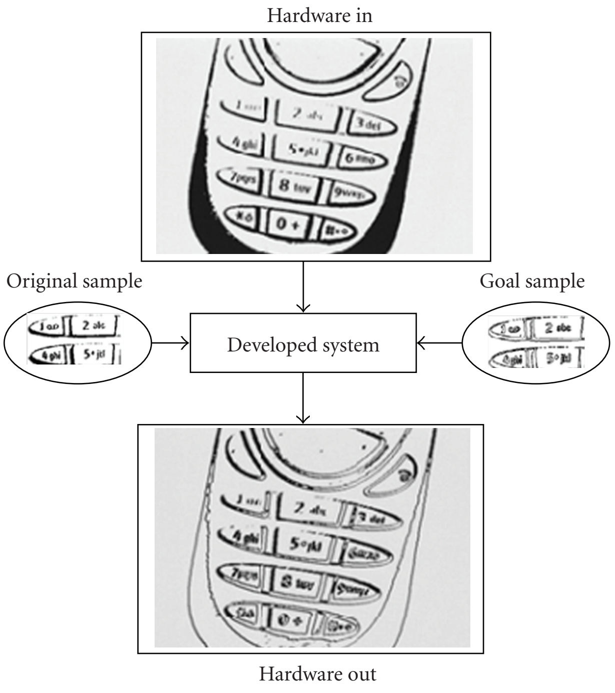

Figure 13: Emulation result of the Photoshop's Trace Contour filter implemented in hardware.

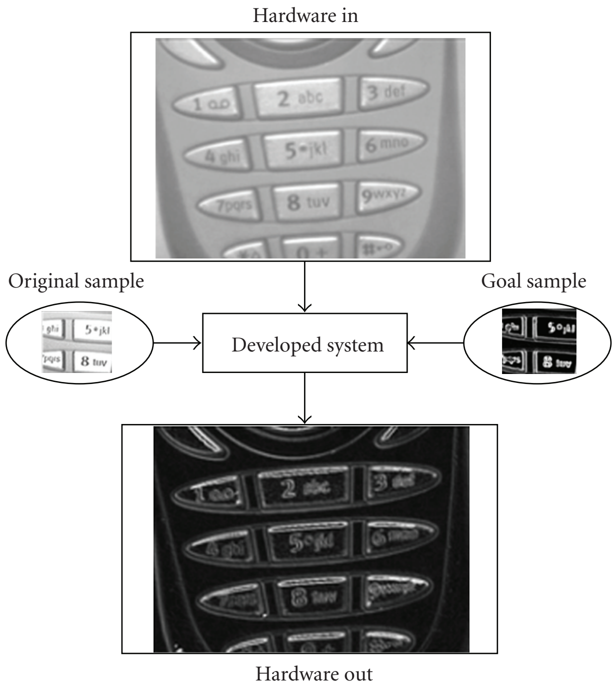

Figure 14: Emulation result of the Photoshop's Glowing Edge filter implemented in hardware.

The training time was less than 10,6 min, and execution time was in real time.

Comparing the results with those obtained from other works in the literature, our implementation presented improvements in fitness, processing time, and programming flexibility. In [20], a genetic algorithm was used for the task of head extraction in music scores. The error found in this work was about 11,8% for a chromosome of size 14. In [23], the error for the same task was greater than 20%. In this paper, the error was less than 0,7% for a chromosome of size 6. In [23] and [20], the processing time of the procedures is not specified. In [13], a genetic algorithm for the task of automatic design of morphological filters is presented. The error found in this work for this task was about 10,59% and the processing time was not performed in real time. In this work, this error was about 2,2% for the same task. In this work, all applications were performed in real time by the developed architecture.

As a contribution of the current paper in relation to the paper [35] presented at SPL, 2010, there are some improvements, such as, the sections were updated, the morphological operators were extended, to gray and color images, new morphological processing arithmetic operators were introduced, additional references were included and new figures and new experiments were shown.

In relation to the paper [36], the current work presents some improvements such as the intelligent reconfiguration of the pipeline architecture by means of a genetic procedure.

Table 2 summarizes all the obtained results, and Table 3 presents the FPGA device used resources.

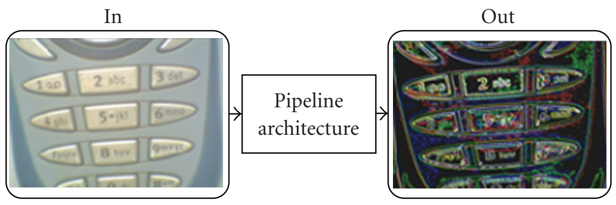

Figure 15: Emulation result of the Photoshop's Glowing Edge filter applied to a color image.

4. Conclusions

In this paper an original reconfigurable architecture using logical and morphological instructions generated automatically by a linear approach based on genetic programming was presented. The developed architecture was based on an FPGA from Altera's Cyclone II family. The system is composed by a 32-stage pipeline and can be used in real-time mathematical morphology and linear applications. The system is able to process 640 × 480 pixels images at 60 frames/sec. Binary, gray-level, and color image practical applications using the developed architecture were presented, and the results were compared with other implementation techniques. The developed system can be applied to digital images for automatic design of nonlinear filters, image segmentation, and pattern recognition. Applications examples were shown where the solutions were expressed in terms of basic morphological operators, dilation, and erosion, in conjunction with arithmetic and logical operators. Compared with other methods described in the literature, the developed methodology presents many improvements in processing time, fitness, and

Table 2: Summary of results.

Example	Generations	Chromosomes	Tree depth	Cross rate	Mut rate	Repr rate	MAE error	Training time	Execution time
Morphological filter	50	25	4	90%	20%	20%	0,4%	4,3 s	Real time
Head extraction	50	50	12	97%	3%	10%	1,1%	12 min	Real time
Trace Contour	50	90	16	97%	3%	10%	2,86%	18 min	Real time
Glowing edge (gray level)	51	70	9	95%	20%	10%	6,2%	10,6 min	Real time
Glowing edge (color)	51	70	9	95%	20%	10%	6,2%	10,6 min	Real time

Table 3: Summary of FPGA device.

Device: Cyclone II EP2C35F672C6/Application	Pins	Memory Bits	LEs (Logic elements)
Morphological Filter	125 (26%)	134208 (28%)	2702 (8%)
Head extraction from music scores	125 (26%)	134208 (28%)	2702 (8%)
Trace contour	125 (26%)	134208 (28%)	2702 (8%)
Glowing edges (níveis de cinza)	125 (26%)	264944 (55%)	5672 (17%)
Glowing edges (colorido)	125 (26%)	233032 (48%)	5249 (16%)

flexibility in relation to program size (variable), types of operators, and extension to color images. The developed method can be used as a guide to morphological design as well as to other applications involving linear image processing.

Acknowledgments

E. C. Pedrino is grateful to the "Fundação de Amparo a Pesquisa do Estado de São Paulo" for the financial support of the project, proc. 2009/17736-4. The authors are also grateful to the Departament of Computer Science, University Federal de São Carlos and to the Department of Electrical Engineering, São Carlos Engineering School, USP, for the support.

References

[1] E. R. Dougherty, Ed., *An Introduction to Morphological Image Processing*, SPIE, Bellingham, Wash, USA, 1992.

[2] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, San Diego, Calif, USA, 1982.

[3] A. R. Weeks Jr., *Fundamentals of Electronic Image Processing*, SPIE, Bellingham, Wash, USA, 1996.

[4] P. Soille, *Morphological Image Analysis, Principles and Applications*, Springer, Berlin, Germany, 1999.

[5] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis and Machine Vision*, Chapman & Hall, Boca Raton, Fla, USA, 1993.

[6] J. Facon, *Morfologia Matemática: Teoria e Exemplos*, Editora Universitária da Pontifícia Universidade Católica do Paraná, Prado Velho, Brazil, 1996.

[7] F. Ortiz, F. Torres, E. De Juan, and N. Cuenca, "Colour mathematical morphology for neural image analysis," *Real-Time Imaging*, vol. 8, no. 6, pp. 455–465, 2002.

[8] P. Maragos, "Lattice image processing: a unification of morphological and fuzzy algebraic systems," *Journal of Mathematical Imaging and Vision*, vol. 22, no. 2, pp. 333–353, 2005.

[9] C. R. Giardina and E. R. Dougherty, *Morphological Methods in Image and Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1988.

[10] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 532–550, 1987.

[11] J. Angulo and J. Serra, *Morphological Coding of Color Images by Vector Connected Filters*, Centre de Morphologie Mathématique, Ecole des Mines de Paris, Paris, France, 2005.

[12] J. Chanussot and P. Lambert, "Total ordering based on space filling curves for multivalued morphology," in *Proceedings of the 4th International Symposium on Mathematical Morphology and Its Applications*, pp. 51–58, 1998.

[13] S. Marshall, N. R. Harvey, and D. Greenhalgh, "Design of morphological filters using genetic algorithms," in *Proceedings of the 10th International Signal Processing Conference (EUSIPCO '00)*, Tampere, Finland, 2000.

[14] M. I. Quintana, R. Poli, and E. Claridge, "Genetic programming for mathematical mor-phology algorithm design on binary images," in *Proceedings of the International Conference of Knowledge Based Computer Systems (KBCS '02)*, pp. 161–171, 2002.

[15] J. Barrera, E. R. Dougherty, and N. S. Tomita, "Automatic programming of binary morphological machines by design of statistically optimal operators in the context of computational learning theory," *Journal of Electronic Imaging*, vol. 6, no. 1, pp. 54–67, 1997.

[16] E. R. Dougherty and R. P. Loce, *Eficient Design Strategies for the Optimal Binary Digital Morphological Filter: Probabilities, Constraints, and Structuring Element Libraries*, Marcel Dekker, New York, NY, USA, 1993.

[17] M. Schmitt, "Mathematical morphology and artificial intelligence: an automatic programming system," *Signal Processing*, vol. 16, no. 4, pp. 389–401, 1989.

[18] J. Barrera, R. Terada, R. Hirata Jr., and N. S. T. Hirata, "Automatic programming of morphological machines by PAC learning," *Fundamenta Informaticae*, vol. 41, no. 1-2, pp. 229–258, 2000.

[19] M. Yu, N. Eua-anant, A. Saudagar, and L. Udpa, "Genetic algorithm approach to image segmentation using morphological operations," in *Proceedings of the International Conference on Image Processing*, pp. 775–779, 1998.

[20] I. Yoda, K. Yamamoto, and H. Yamada, "Automatic acquisition of hierarchical mathematical morphology procedures by genetic algorithms," *Image and Vision Computing*, vol. 17, no. 10, pp. 749–760, 1999.

[21] J. Bala and H. Wechsler, "Shape analysis using morphological processing and genetic Algorithms," in *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (TAI '91)*, pp. 130–137, Los Alamitos, Calif, USA, 1991.

[22] N. R. Harvey and S. Marshall, "The use of genetic algorithms in morphological filter design," *Signal Processing: Image Communication*, vol. 8, no. 1, pp. 55–71, 1996.

[23] M. I. Quintana Hernandez, *Genetic programming applied to morphological image processing*, Ph.D. thesis, School of Computer Science, University of Birmingham, 2005.

[24] D. Barrios, A. Carrascal, D. Manrique, and J. Ríos, "Optimisation with real-coded genetic algorithms based on mathematical morphology," *International Journal of Computer Mathematics*, vol. 80, no. 3, pp. 275–293, 2003.

[25] D. Barrios, D. Manrique, and J. Porras, "Real-coded genetic algorithms based on ma-thematical morphology," *Advances in Pattern Recognition*, vol. 1876/2000, pp. 706–715, 2000.

[26] T. Belpaeme, "Evolution of visual feature detectors," in *Proceedings of the 1st European Workshop on Evolutionary Computation in Image Analysis and Signal Processing (EvoISAP '99)*, R. Poli, S. Cagnoni, H. M. Voigt, T. Fogarty, and P. Nordin, Eds., pp. 1–10, Goteborg, Sweden, 1999.

[27] J. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, Mass, USA, 1975.

[28] J. Koza, *Genetic Programming*, MIT Press, Cambridge, Mass, USA, 1992.

[29] GP FAQ, 2002, http://www.cs.ucl.ac.uk/research/genprog/gp2faq/gp2faq.html.

[30] M. Bhattacharya, A. Abraham, and B. Nath, "A linear genetic programming approach for modeling electricity demand prediction in victoria," in *Proceedings of the hybrid information systems (HIS '01)*, pp. 379–393, 2001.

[31] M. Oltean, C. Groşan, and M. Oltean, "Encoding multiple solutions in a linear genetic programming chromosome," in *International Conference on Computational Science*, pp. 1281–1288, 2004.

[32] D. Goldberg and K. Déb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*, G. Rawlins, Ed., pp. 69–93, Morgan Kauffmann, Boston, Mass, USA, 1991.

[33] Altera, 2008, http://www.altera.com/education/univ/materials/boards/unv-de2-board.html.

[34] E. C. Pedrino and V. O. Roda, "Pipeline architecture for real time morphological color image processing," in *Proceedings of the 2nd Southern Conference on Programmable Logic, Mar Del Plata, Argentina, Fpga Based Systems*, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain, 2006.

[35] E. C. Pedrino, J. H. Saito, and V. O. Roda, "Architecture for binary mathematical mor-phology reconfigurable by genetic programming," in *Proceedings of the 6th Southern Conference on Programmable Logic*, Universidade Federal de Pernambuco, Porto de Galinhas, Brasil, 2010.

[36] E. C. Pedrino and V. O. Roda, "Real-time morphological pipeline architecture using high-capacity programmable logical devices," *Journal of Electronic Imaging*, vol. 16, no. 2, Article ID 023002, 2007.