An Optimized Architecture for Implementing Image Convolution with Reconfigurable Hardware

Miguel A. Vega-Rodríguez, Juan M. Sánchez-Pérez, Juan A. Gómez-Pulido

Univ. de Extremadura. Dept. de Informática
Escuela Politécnica. Campus Universitario, s/n. 10071 Cáceres. Spain
{mavega,sanperez,jangomez}@unex.es FAX:+34-927-257202

Abstract. Convolution is a very important operation within artificial vision. It can be characterized as being computationally intensive, so it is hard to implement real-time convolution. One reason for this is the vast amount of data that requires processing (more than nine million pixels per second for typical image sources). This paper introduces a new architecture along with its optimizations for implementing convolution operations in FPGAs. The proposed architecture uses techniques of parallelism with some improvements. The system is based on the HOT2-XL PCI board, and we have developed a Visual C++ application to validate our hardware designs. This environment is based on a library of hardware modules implementing the most common operations in image processing. In this paper we focus on the convolution modules. The found results illustrate the effectiveness of our improvements allowing real-time processing, a minimum resource use and a high operation frequency.

1 Introduction

Convolution is a popular operation in the field of image processing, and it has been used in a great number of applications [1], [2], [3]. Among these applications, its use for artificial vision highlights. The main challenge is that the artificial vision systems are usually used in real-time applications (30 images/second). For this reason, at present, the implementation of convolution by means of reconfigurable hardware is being investigated [4]. Some proposals even use techniques like Simulated Annealing and Genetic Programming in order to find the best implementation for convolution [5].

Reconfigurable computing systems [6], [7] are very flexible, allowing that new operations can be implemented in the existent hardware, and they offer enough speed for the execution in real time, which is not attainable by software many times. What is more, the price/performance ratio of these systems makes them a broadly competitive alternative to ASICs [8].

Very diverse FPGA-based boards are appearing in the market for reconfigurable computing. These boards have very different communication interfaces with the host: parallel port, serial port, USB, Ethernet,... But in general, the boards devoted to image processing in real time have a PCI (Peripheral Component Interconnect) interface. The main reason is that it gives them the necessary speed to work as coprocessors.

Also, PCI bus has a growing popularity for image processing due to its: high bandwidth, independence of the CPU, plug-n-play, configurability and expandability, standardization, and master-slave behaviour [9], [10].

In conclusion, FPGAs [11], [12] in these boards have 32-bit buses for communicating with the external world, either for their communication through the PCI bus or for accessing to the RAM memory that usually comes integrated in the boards. Therefore, the images to be processed by the FPGA will arrive to it through a bus of 32 bits, and this has a great influence on the hardware architecture for implementing the convolution. An example of this type of boards is the HOT2-XL board [13], [14] of VCC [15] that has been used to carry out our experiments.

This work is organized as follows. In section 2 the convolution operation is described shortly. The following section presents the proposed architecture for implementing the convolution using FPGAs. Section 4 explains some improvements performed. Then, in section 5, we show and analyze the implementation results. Finally, the conclusions are presented.

2 Convolution Operation

Convolution is an operation of linear spatial filtering. Figure 1 illustrates the convolution concept using a 3x3 mask. The shading window will go moving through the whole image I to produce the corresponding pixels of the image O, keeping in mind that each localization of the window will generate only one pixel in the image O.

Fig. 1. Convolution concept based on a 3x3 kernel

For this operation, to apply the mask on a given position means to multiply each weight of the mask by the pixel of the image on which is, and to add the resulting products. This sum is used as the value of the pixel for the output image. Therefore, the convolution of an image I with a square mask w of width n=2k+1 will generate an image O that could be defined by means of the weighted average of figure 2. Generally, n=3 (k=1) is taken. Furthermore, according to the results of Manseur and Wilson [16], any other mask can be decomposed in a chain of masks of dimensions 3x3. The mask weights determine the action of spatial filtering to carry out, being able to be a low-pass, high-pass, gradient, laplacian,... filter.

$$O(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot w(i,j)$$

Convolution Mask w

j \ i	-1	0	1
-1	A	B	C
0	D	E	F
1	G	H	I

Fig. 2. Definition of convolution and example of 3x3 mask

3 Proposed Architecture

As the data bus for communicating the FPGA with the external world is a 32-bit bus, in each read/write operation four pixels of an image (supposing 8-bit pixels) are obtained/sent. To gain benefit from this situation the proposed hardware architecture replicates the functional units in order to apply the convolution operation simultaneously on four pixel neighbourhoods. In this way, it takes advantage of the inherent neighbourhood parallelism in the convolution. A first approach for the simultaneous computation of these four output pixels would be the one shown in figure 3. The images are divided in pixels (each square) grouped in words of 32 bits (4 pixels of 8 bits). For more clarity each application of the mask has been represented with a different texture.

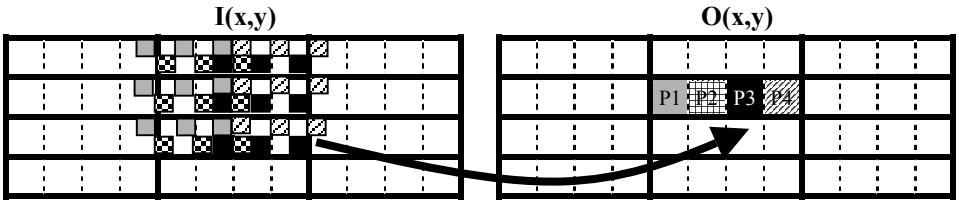


Fig. 3. First approach to compute four pixels simultaneously in the resulting image

However, this approach is not good since it implies nine read operations to obtain the nine necessary words of the input image. In order to reduce the number of read operations, and therefore increasing the performance of the convolution operation, the approach of figure 3 has been modified as it is illustrated in figure 4.

As we see, this approach only forces to read six words of the input image. However, pipelining this approach in a pipeline with two stages it is possible to get an operation outline that writes four pixels in the output image in each clock cycle, only reading three words of the input image by cycle (see figure 5).

I(x,y) refers to the input image, and O(x,y) to the output image. Each Filter circuit computes the associate resulting pixel (P1, P2, P3 or P4) after applying the corresponding convolution operation. In each clock cycle, three words of the input image are read (the three necessary rows), and a new word with four resulting pixels is written. The computation of these four pixels is carried out in two stages (pipelining), also marked in figure 5. In stage 1 the resulting pixels P1, P2 and P3 are computed,

storing them in registers that will be used in stage 2. In stage 1 the columns used to compute the pixel P4 in stage 2 are also stored in registers. Finally, in stage 2, the word (with its four pixels Pi) is written in the output image.

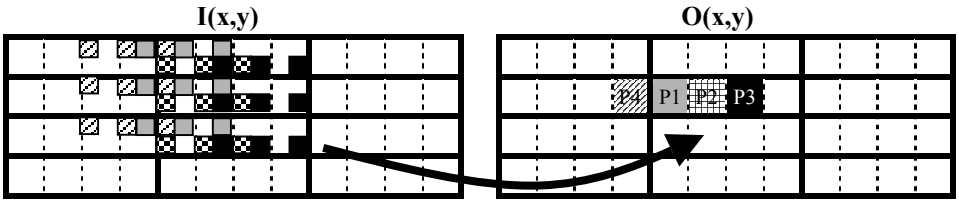


Fig. 4. Approach followed to compute four pixels simultaneously in the resulting image

Thanks to the pipelining both stages operate at the same time, improving their performance. While stage 1 computes the pixels P1, P2 and P3 according to the three words read in the current clock cycle, stage 2 computes P4 and writes the resulting word (P1, P2, P3 and P4) associated with the three words read in the previous cycle.

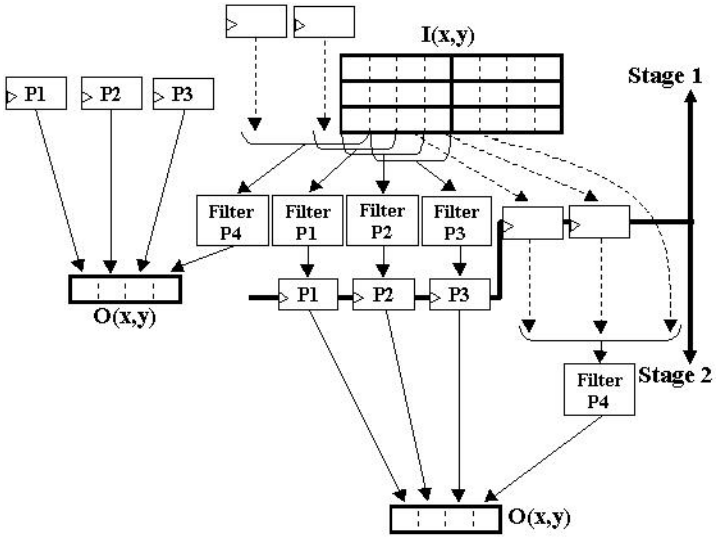


Fig. 5. Pipelining of the approach presented in figure 4

4 Optimizations on each Filter Circuit

On each Filter circuit (figure 5) we have carried out two important improvements. On the one hand, the multipliers have been optimized to use less resources. And on the other hand, the datapaths have been optimized to use less bits in each operation and connection. Therefore, the system performance is improved and the required resources are reduced. Furthermore, all this leads to a more regular architecture that will allow us to reuse resources.

As an example to explain the optimizations carried out we study which would be the resulting circuit to implement a laplacian filter, whose more popular mask is in figure 6.

$$W = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Fig. 6. Mask habitually used for a laplacian filter

In this case each Filter circuit of figure 5 could be implemented like it is shown in figure 7 [17]. In this figure we observe the necessary nine multipliers and the summing network (with eight adders). A circuit to control the range of the resulting pixel could be added, that is, to make sure that the resulting pixel has a value between 0 and 255.

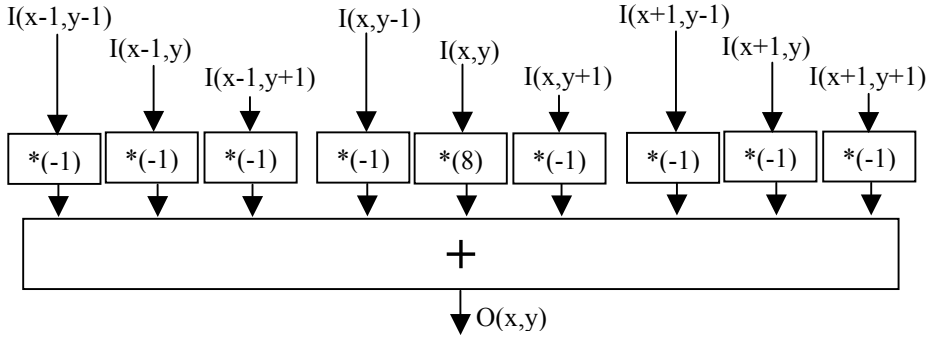
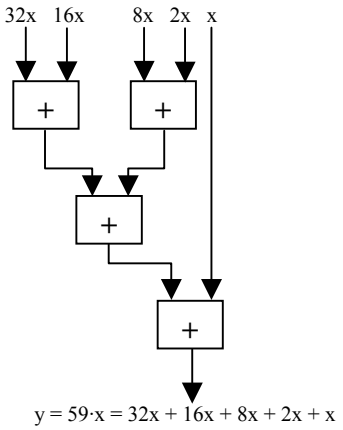


Fig. 7. Possible implementation of each Filter circuit of figure 5

The optimization of the multipliers in figure 7 is based on the use of constant multiplication weights, which is habitual in artificial vision. This optimization can reduce significantly the required resources, and increase the performance of an implementation. Let us consider the implementation of the operation "y = W·x". If the weight W is constant we can decompose it in an adder tree according to its binary representation. Figure 8(a) shows the result of applying this technique to the previous product with W=59 (111011 in binary code). The products that are power of two can be implemented by means of shifts, or even more, by means of the use of connections with zeros (direct connections to ground).

In order to treat the negative weights, the convolution mask is broken down in a positive mask (with the positive weights) and a negative one; A similar division is followed by Lisa [18]. Each of these masks is processed in parallel, and finally the output values are subtracted (see figure 8(b) for the laplacian case). In this case both masks only have weights that are power of two, so their implementation is immediate. Figure 9 illustrates the resulting Filter circuit after this optimization (the number of bits in each adder and connection is also indicated).



$$W = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Fig. 8. (a) Example of decomposition of a multiplication in an adder tree. (b) Decomposition of the mask of figure 6 into two masks

However, in our final architecture the multipliers have not only been optimized, but the datapaths have also been optimized with the purpose of using less bits in each operation and connection, and therefore, to improve the system performance and to reduce the necessary resources; as well as obtaining a more regular architecture.

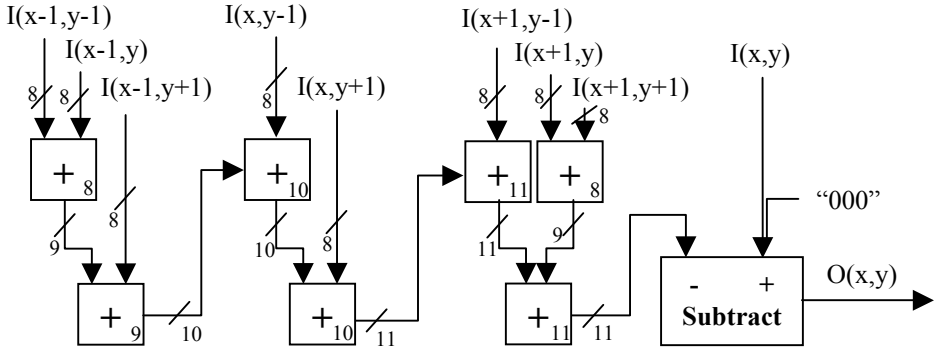


Fig. 9. Implementation of a laplacian filter after applying the optimization of the multipliers

Figure 10 shows the resulting circuit for the laplacian filter, after optimizing the datapaths in the circuit of figure 9. In this figure the operations and connections that have been reduced in the number of necessary bits after applying the optimization appear in shady color. Evidently, both optimizations (multipliers and datapaths) will be of more importance when the convolution to implement is more complex.

Thanks to the great gotten regularity, it is still possible to apply a new optimization, reusing the intermediate results obtained in the adder tree of the different Filter circuits. As we have explained, the final circuit would have a total of four Filter circuits, operating in parallel, with an architecture like the one shown in figure 10. However, as these four circuits have many common inputs (of the nine pixels implied in a 3x3 convolution up to six are the same in two different Filter circuits), it is possible to reuse many partial sums, therefore, saving a great quantity of adders. Figure 11 illustrates this last optimization.

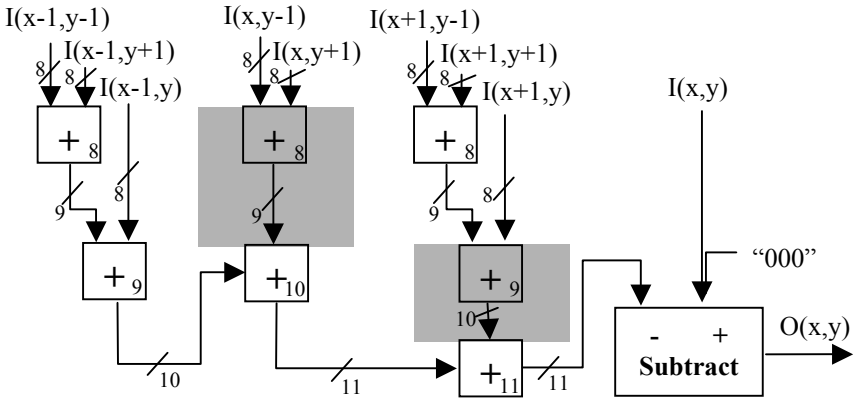


Fig. 10. Implementation of a laplacian filter after applying the datapath optimization

In this figure the four Filter circuits are shown together, sharing resources. P1, P2, P3 and P4 are the four pixels to obtain in each clock cycle according to the execution outline explained in figures 4 and 5, so that P1, P2 and P3 should be stored temporarily in registers to get the pipelining of the convolution. Figure 12 illustrates the diagram for anyone of the four final circuits of figure 11 used to obtain each pixel.

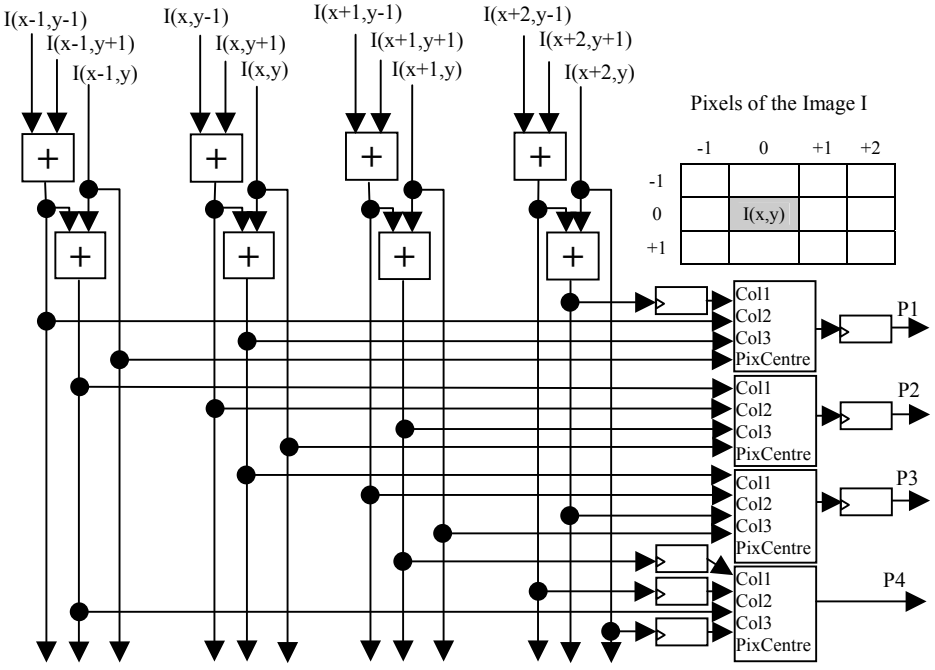


Fig. 11. Convolution with a laplacian filter, reusing resources in the four Filter circuits

Each input Col_i makes reference to the sum of a column (keeping in mind that in the central column, Col2, only the inferior and superior pixels are added) and PixCentre is the pixel on which the convolution mask is placed. Analyzing figure 11 we

conclude that 8 adders are only necessary to compute the sums for columns (Col_i). If we had repeated the diagram of figure 10 four times, a total of 20 adders would have been needed to carry out these same sums for columns; therefore, this optimization has lead to an important saving of 12 adders (of eight or more bits). Even more, according to figure 5 it is necessary the use of a great quantity of registers to store 6 pixels of the image, which are required to compute P4 and P1. Figure 11 also shows that those 6 registers can decrease to 4, storing the results of the partial sums among them instead of the 6 individual pixels.

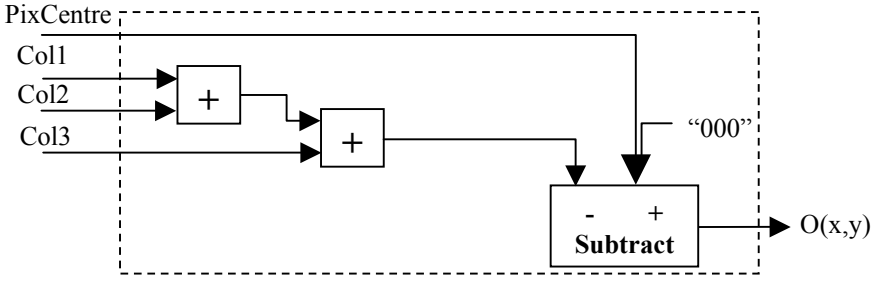


Fig. 12. Internal diagram for anyone of the four final circuits of figure 11

5 Experimental Results

We have implemented diverse convolution filters following the proposed architecture and the presented optimizations. All of them have been implemented on the XC4062XLA-09HQ240C FPGA [19] included in the HOT2-XL PCI board. This board has been placed in a PCI slot of a PC, creating a CCM [20]. The experiments have been carried out by means of a Windows Visual C++ application. This application manages the system and reconfigures the HOT2 board (its FPGA) with different hardware modules as it is necessary. Table 1 shows the results obtained for the different convolution filters.

Table 1. Experimental results for different convolution filters

Operation	Aver. Exec. Time (ms)	Resource Use (CLBs)	Max. Freq. (MHz)	Minimum Period (ns)
Vertical Gradient Filter	581.016	65 (2.82%)	31.426	31.821
Horizontal Gradient Filter	718.125	105 (4.56%)	32.633	30.644
Diagonal Gradient Filter	717.734	109 (4.73%)	32.449	30.818
High-Pass Filter	868.516	322 (13.98%)	32.853	30.439
Low-Pass Filter	867.734	238 (10.33%)	30.000	33.333
Laplacian Filter	867.344	290 (12.59%)	33.638	29.728

The second column shows the average execution time in ms for each operation on 30 images of 640x480 pixels with 256 gray levels. With a 16-MHz clock, in the HOT2-XL board, the time to process 30 images ranges from 581.016 to 868.516 ms, according to the operation. In these times, the reconfiguration time of the board (FPGA) is included. The board has an average reconfiguration time of 440 ms. Notice

you that if we use the same operation repeatedly, the board would only be configured the first time. Furthermore, the clock could be changed to higher frequencies, inside the limit indicated in the fourth column, if it was necessary to obtain a larger performance.

The overload due to the reconfiguration time also has great importance. Our system uses the Configuration Cache included in the HOT2-XL to alleviate this overload, using a LRU (Least Recently Used) replacement policy. If a hardware module was already loaded in the Configuration Cache the average reconfiguration time would decrease to 270 ms. Therefore, the most recent operations would have a significantly lower average execution time to the one shown in the table. We can observe that all the developed hardware modules allow real-time processing, 30 images per second.

In the third column of table 1 we show the resource use of the XC4062XLA-09HQ240C FPGA for our hardware modules. This use is expressed by number of CLBs (Configurable Logic Blocks), knowing that the XC4062XLA FPGA has a total of 2304 CLBs. The percentage of CLBs used of those 2304 is also indicated. The quantities presented in this column and the following ones have been obtained from the reports generated by the Xilinx Foundation Series tools [21] after the synthesis of each hardware module. Therefore, they are real measurements on implementations already carried out, and not estimations.

It should be kept in mind that each module has its functional units replicated four times to take advantage of the HOT2-XL board architecture [13]. This fact multiplies the use of resources by four. Even more, each module also includes the necessary circuitry for the internal management of the on-board memory. As we see, the designed hardware modules present an efficient use of the FPGA resources, requiring in the worst case a number of CLBs lesser than 14%.

The fourth column of table 1 presents the maximum frequency admitted by each hardware module. To facilitate the data interpretation, the minimum clock period is also offered. From the table we conclude that all the hardware modules admit a maximum clock frequency about 33 MHz, coinciding with the PCI bus specifications for a 32 bit/33 MHz system, and therefore, a 132 Mbytes/sec. bandwidth [9].

6 Conclusions

To configure the HOT2-XL PCI board we are developing a library devoted to image processing. At present, the library consists of 16 hardware modules of very diverse types: point, histogram, convolution, mathematical morphology,... operations [22]. In this work we have realized a study of the architecture and optimizations proposed for implementing in an FPGA the convolution operations.

The practical results illustrate the effectiveness of the presented architecture and improvements: use of parallelism techniques like replication and pipelining, optimization of the multipliers by means of adder trees, optimization of the datapaths, search for a high regularity, reutilization of common resources (adders), etc. Everything has allowed us to get real-time processing, a minimum use of resources and a high frequency of operation.

References

1. BURDICK, H.E.: 'Digital imaging. Theory and applications' (McGraw-Hill, 1997)
2. DOUGHERTY, E.R.; LAPLAVE, P.A.: 'Introduct. to real-time imaging' (SPIE Press, 1995)
3. HARALICK, R.M.; SHAPIRO, L.G.: 'Computer and robot vision' (Addison-Wesley, Massachusetts, 1992, vol. 1)
4. JAMRO, E.; WIATR, K.: 'FPGA Implementation of Addition as a Part of the Convolution'. EuroMicro Symposium on Digital System Design, DSD'2001, IEEE Computer Society, September 2001, Warsaw, Poland, pp. 458-465
5. JAMRO, E.; WIATR, K.: 'Genetic Programming in FPGA Implementation of Addition as a Part of the Convolution'. EuroMicro Symposium on Digital System Design, DSD'2001, IEEE Computer Society, September 2001, Warsaw, Poland, pp. 466-473
6. MANGIONE-SMITH, W.H.; HUTCHINGS, B.L.: 'Configurable computing: the road ahead'. Reconfigurable Architectures Workshop, RAW'97, 1997, Geneva
7. VILLASENOR, J.; MANGIONE-SMITH, W.H.: 'Configurable computing', Scientific American, 1997, 276(6), pp. 54-59
8. SMITH, M.J.S.: 'Application-specific integrated circuits' (Addison-Wesley, 1997)
9. PCI SPECIAL INTEREST GROUP: 'PCI local bus specification - Revision 2.1'. http://www.pcisig.com, June 1995
10. SHANLEY, T.; ANDERSON, D.: 'PCI system architecture' (Addison-Wesley, 1995)
11. BROWN, S.; FRANCIS, R.J.; ROSE, J.; VRANESI, Z.G.: 'Field-programmable gate arrays' (Kluwer Academic Publishers, 1992)
12. BROWN, S.; ROSE, J.: 'FPGA and CLPD architectures: a tutorial'. Proc. of the IEEE Design & Test of Computers, 1996, 13(2), pp. 42-57
13. VIRTUAL COMPUTER CORPORATION: 'H.O.T. II architecture'. Technical Bulletin, Virtual Computer Corporation, October 1998
14. VIRTUAL COMPUTER CORPORATION: 'H.O.T. II hardware guide. Version 2.0'. Virtual Computer Corporation, 1999
15. VIRTUAL COMPUTER CORPORATION: http://www.vcc.com, 2002
16. MANSEUR, Z.Z.; WILSON, D.C.: 'Decomposition methods for convolution operators'. CVIP: Graphical Models and Image Processing, 1991, vol. 53, pp. 428-434
17. BAXES, G.A.: 'Digital image processing. Principles & applic.' (John Wiley&Sons, 1994)
18. LISA, F.: 'Configurable computing for real-time vision'. Doctoral Thesis, Dept. of Computer Sciences, Autonomous University of Barcelona, Bellaterra, Spain, September 1998
19. XILINX INC.: 'The programmable logic data book'. Xilinx Inc., http://www.xilinx.com/partinfo/databook.htm, 2002
20. BUELL, D.A.; POCEK, K.L.: 'Custom computing machines: an introduction'. Journal on Supercomput, 1995, nº 9, pp. 219-230
21. XILINX INC.: http://www.xilinx.com, 2002
22. VEGA, M.A.; SÁNCHEZ, J.M.; GÓMEZ, J.A.: 'Real-Time Image Processing with Reconfigurable Hardware'. The 8th IEEE International Conference on Electronics, Circuits and Systems, ICECS'2001, IEEE Computer Society, September 2001, Malta, pp. 213-216