Proceedings of the 10th Mediterranean Conference
on Control and Automation - MED2002
Lisbon, Portugal, July 9-12, 2002.

AN FPGA-BASED IMPLEMENTATION FOR MEDIAN FILTER MEETING THE REAL-TIME REQUIREMENTS OF AUTOMATED VISUAL INSPECTION SYSTEMS

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

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

Keywords: Factory Automation, Image Processing, Median Filter, Reconfigurable Hardware, Real-Time Processing.

Abstract

Image processing is a very important field within factory automation, and more concretely, in the automated visual inspection. The main challenge normally is the requirement of real-time results. On the other hand, in many of these applications, the existence of impulsive noise in the acquired images is one of the most habitual problems. Median filter is a robust method to remove the impulsive noise from an image. It is a computationally intensive operation, so it is hard to implement it in real time. This paper introduces a new architecture and optimizations for its implementation with FPGAs. The practical results show the effectiveness of our improvements allowing real-time processing and a minimum use of resources.

1 Introduction

Image processing is a very important field within industrial automation, and more concretely, in the automated visual inspection. For example, automatically analyzing predetermined features of manufactured parts on an assembly line to look for defects and process variations [1]. In these applications, the main challenge normally is the requirement of real-time results.

On the other hand, in many of these applications, the acquired images must pass through a stage of image preprocessing in order to remove distracting and useless information from the images. For example, the existence of impulsive noise in the images is one of the most habitual problems.

Median filter is the nonlinear filter more used to remove the impulsive noise from an image [4, 1]. It is a more robust method than the traditional linear filtering, because it preserves the sharp edges, although it also has a much higher computational cost. When the median filter must be carried out in real time, the software implementation in general-purpose processors does not usually give good results. ASICs [9] have been required traditionally. However, they imply a limited functionality due to their predefined architecture. Also, the price for application as well as the production time are usually prohibitive.

Reconfigurable computing architectures [12] are sufficiently flexible so that new operations can be implemented in the existent hardware, and they are quite quick for real-time execution. Moreover, the price/performance ratio of these systems makes them a broadly competitive alternative to ASICs. FPGAs [2] have been identified as the natural platform for CCMs [3] due to their reprogrammability [5].

In section 2 the median filter is described briefly. Section 3 presents the architecture that we propose for implementing the median filter by means of FPGAs, while section 4 illustrates the optimizations performed in this architecture. Then, in the following section, we show and analyze the

experimental results; presenting the conclusions in section 6.

2 The Median Filter

Median filter is a spatial filtering operation, so it uses a 2-D mask that is applied to each pixel in the input image. To apply the mask means to centre it in a pixel, evaluating the covered pixel brightnesses and determining which brightness value is the median value. The median value is determined by placing the brightnesses in ascending order and selecting the centre value [1]. The obtained median value will be the value for that pixel in the output image. Figure 1 shows an example.

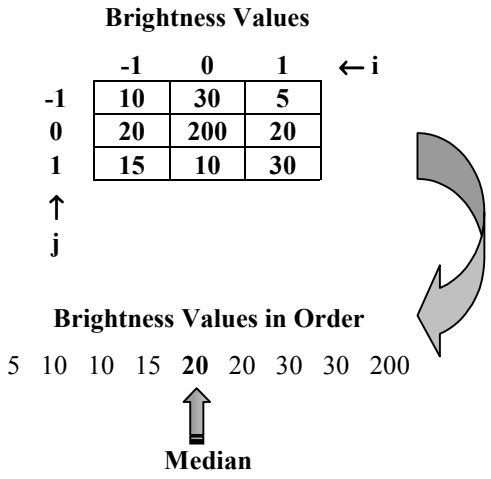

Figure 1: Application of the median filter.

Habitually a 3x3 median filter is used, since bigger filters usually eliminate small edges. We have therefore focused on the 3x3 median filter implementation.

3 Architecture for the Median Filter

Very diverse FPGA-based custom-computing boards are appearing in the market. These boards possess different interfaces for their communication with the host. But in general, boards devoted to real-time image processing have a PCI interface, because it gives them the necessary speed to work as coprocessors. Also, PCI bus has a growing popularity due to its interesting properties [7]. In conclusion, FPGAs in these boards have 32-bit buses for their communication with the external world, either for their communication through PCI bus or for the access to the local RAM that the boards usually have. An example of this board type is the HOT2-XL board [13] of Virtual Computing Corporation that has been used to carry out our experiments.

The fact that we have a 32-bit data bus has a very large influence in the necessary hardware architecture for implementing image processing operations, because it causes that in each read/write operation we obtain/send four image pixels (supposing 8-bit pixels). We have gained benefit from this situation replicating the functional units in order to apply the median filter simultaneously on four pixel neighbourhoods. In this way we take advantage of the inherent neighbourhood parallelism, and we accelerate the operation four times. Figure 2 presents the approach followed for the simultaneous computation of these four output pixels.

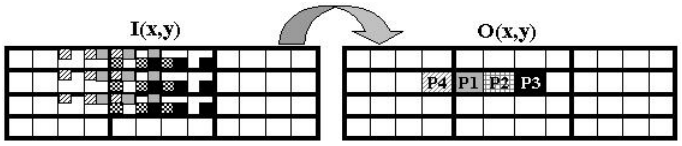

Figure 2: Computation of 4 pixels simultaneously.

Images are divided in pixels (squares) that are grouped in 32-bit words (4 pixels). The value of each output pixel O(x,y) is computed using the 9 pixels of the image I that are inside the 3x3 mask with centre in I(x,y). Each mask application has been represented with a different texture. Note that the pixel P4 of the previous word is computed and not that of the current word. In this way, it is only necessary to read six words in the input image instead of nine, reducing the number of read operations, and therefore increasing the performance. Pipelining this approach using two stages it is possible to get an architecture that writes four pixels (one word) in the output image in each clock cycle, only reading three input image words by cycle (Figure 3).

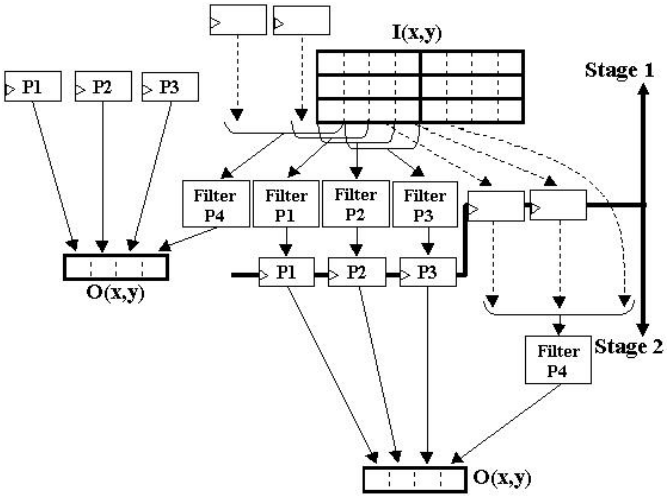

Figure 3: Pipelining of the followed approach.

Each Filter circuit computes the associate resulting pixel (P1, P2, P3 or P4) as the median of the nine implicated pixels. The computation of these four pixels is carried out in two stages (pipelining). In the first stage the resulting pixels P1, P2 and P3 are computed, storing them in registers to use them in stage 2. In stage 1 the columns required to compute the pixel P4 in stage 2 are also stored in registers. It is also in stage 2 when the word (with its four pixels Pi) is written in the output image. Thanks to the pipelining both stages operate at the same time, gainning in 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 associated with the three words read in the previous cycle.

4 Proposed Architecture Optimization

Figure 4 shows the classic approach for sorting 9 pixels and selecting the median. Each Filter circuit of Figure 3 could be implemented with this approach. 41 basic nodes are necessary in this sorting network.

This approach has suffered several proposals of optimization, reducing the number of necessary nodes. Among them we highlight the one presented by Morcego et al. [6] (Figure 5) with a sorting network composed of 27 basic nodes.

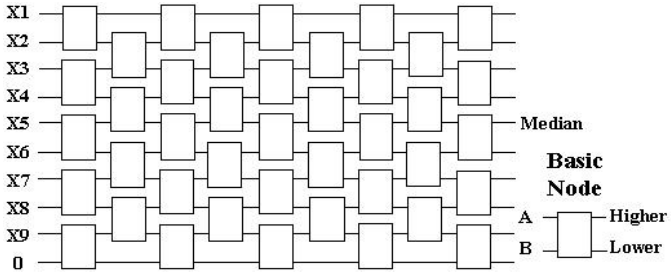

Figure 4: Classic systolic array for sorting 9 pixels.

We have followed the scheme shown in Figure 6, which is similar to the one proposed by Smith [8], and much better than the one presented in Figure 5, since it needs a very lower number of basic nodes (19 instead of 27). This scheme uses the minimum exchange network required to produce the median from nine pixels by performing a partial sorting.

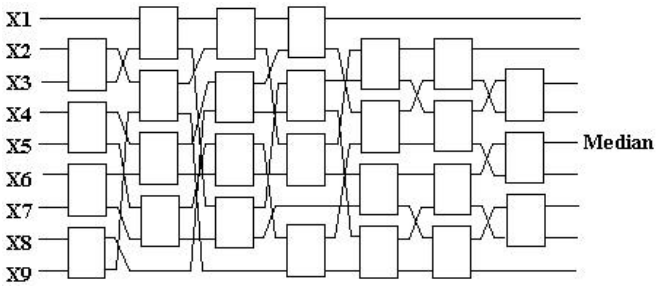

Figure 5: Optimized systolic array.

Each basic node allows to sort two elements. To do that, each node compares the two elements by means of an 8-bit comparator, using its output in two 2:1 multiplexers (see Figure 7).

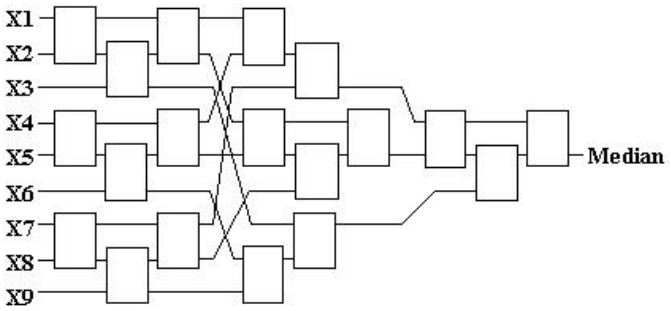

Figure 6: Implementation for each Filter circuit.

In Figure 6, the triangular groups with three nodes perform a full sorting on three elements. In this

figure we observe that some nodes only use one of their two outputs. This allows to eliminate in these nodes one of the multiplexers, reducing the used resources. In total, 8 nodes use only one output, so 8 multiplexers can be saved. This scheme is repeated four times because four Filter circuits exist (Figure 3), so the total number of saved multiplexers is 32.

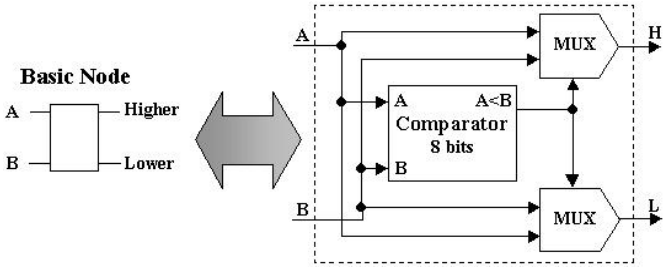

Figure 7: Scheme for each basic node.

A new optimization is applied reusing the intermediate results obtained in the node tree of the different Filter circuits. As we have explained, the final circuit have a total of four Filter circuits, operating in parallel, with the architecture shown in Figure 6. However, as these four circuits have many common inputs (of the 9 implicated pixels up to six are equal in two different Filter circuits), it is possible to reuse many of the partial sortings, therefore, saving a great quantity of nodes. Figure 8 shows this last optimization.

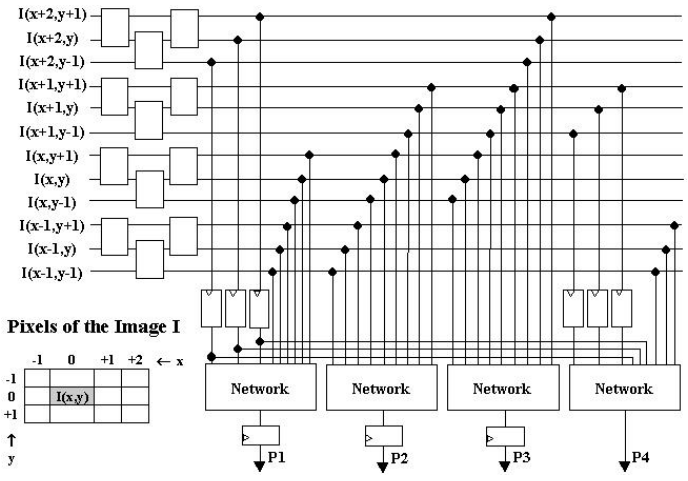

Figure 8: Scheme for the median filter reusing resources in the four Filter circuits.

The four Filter circuits are shown together, sharing resources. P1, P2, P3 and P4 are the four pixels obtained in each clock cycle according to the execution scheme of Figures 2 and 3, so that P1, P2 and P3 should be stored temporarily in registers to get the pipelining of the operation. Figure 9 illustrates the internal scheme for anyone of the Network circuits.

Each input CiL makes reference to the lower value of the column i, each CiH to the higher value and CiM to the middle one. The basic nodes that only use one of their two outputs have been simplified, removing one of their two multiplexers. So the quantity of required resources is reduced. In total, eight nodes of Figure 9 only use one output, so 8 multiplexers can be saved. As this scheme is repeated four times since there are four Network circuits, the total number of saved multiplexers is 32.

Analyzing Figure 8 we conclude that 12 basic nodes are only necessary to compute CiL, CiM and CiH. If we repeated the scheme of Figure 6 four times, 36 nodes had been needed to perform the same sortings. The resource reusing therefore saves 24 nodes, that is to say, 24 comparators of 8 bits and 48 multiplexers.

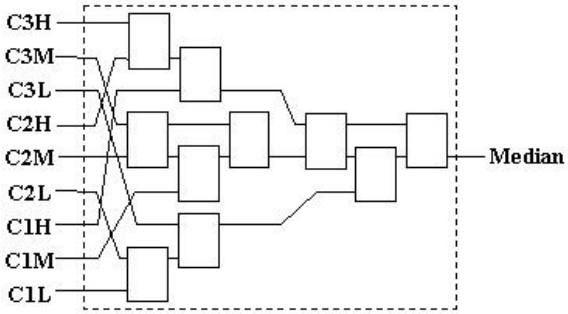

Figure 9: Internal scheme for each Network circuit.

In conclusion, thanks to the applied optimizations we have gotten the reduction in resources shown in Table 1.

Optimization	Comments	Reduction of Resources
Reduction of the sorting and selection network (figure 6 instead of figure 4)	22 basic nodes by Filter circuit (4) = 88 basic nodes	88 comparators of 8 bits and 176 2:1 multiplexers
Reuse of common resources (intermediate results, figure 8)	24 basic nodes	24 comparators of 8 bits and 48 2:1 multiplexers
Optimization of the basic nodes removing one output	8 multiplexers by Network circuit (4) = 32 multiplexers	32 2:1 multiplexers
Totals	---	112 comparators of 8 bits and 256 2:1 multiplexers

Table 1: Optimizations applied to the proposed architecture.

5 Experimental Results

We have implemented the median filter, following the presented architecture and optimizations, on the XC4062XLA-09HQ240C FPGA of the HOT2-XL board. This board has been placed in a PCI slot of a PC, creating a CCM [3]. The experiments have been performed by a Windows Visual C++ application that manages the system and reconfigures the board (its FPGA) with different hardware modules when it is necessary. In this application, the image processing operations can be carried out by hardware (FPGA) or software (Visual C++), allowing the result comparison. Table 2 shows the obtained results.

Average Execution Time for SW (ms)	**84459.766**
Average Execution Time for HW (ms)	**998.203**
Use of Resources (CLBs)	**634 (27.52%)**
Maximum Frecuency (MHz)	**16.162**
Minimum Period (ns)	**61.873**

Table 2: Experimental results for the median filter.

The first and second row show a comparison between the hardware implementation and its software version. In both rows, the average execution time is indicated for the operation on 30 images of 640x480 pixels of 256 gray levels. With a 16-MHz clock, in the HOT2-XL board, the time to process 30 images is 998.203 ms. The reconfiguration time of the board, 440 ms, is included in this time. Notice that if we use the same operation repeatedly, the board would only be configured the first time.

The software version, written in Visual C++ and compiled with the appropriate optimizations, requires 84459.766 ms in a PC with a 350-MHz Pentium-II processor and 64 Mb of RAM. Therefore, the hardware implementation represents an important performance improvement, obtaining a time 85 times smaller. Furthermore, note that the HOT2-XL is running at a clock rate approximately 1/22 (16 MHz) that of the microprocessor (350 MHz).

The overload due to the reconfiguration time has great importance. For this reason, we have used the Configuration Cache included in the HOT2-XL [13] for alleviating this overload. If a hardware module was already loaded in the Configuration Cache the average reconfiguration time would decrease to 270 ms. Therefore, the median filter would have in its hardware version a significantly lower average execution time than the one shown.

If we intend to process images in real time (30 images/second), and we suppose that one processing operation by image is only applied, we must admit a maximum time of 1000 ms. Table 2 shows that the software version can not reach this speed. However, this is possible using our hardware module.

In the third row we show the FPGA resource use in number of CLBs (Configurable Logic Blocks), knowing that the XC4062XLA FPGA has 2304 CLBs. The percentage of CLBs used of those 2304 is also indicated. The quantities presented in this one and the following rows have been obtained from the reports generated by the Xilinx Foundation Series tools after the hardware module synthesis. Therefore, they are real measurements on the implementation already carried out, and not estimations.

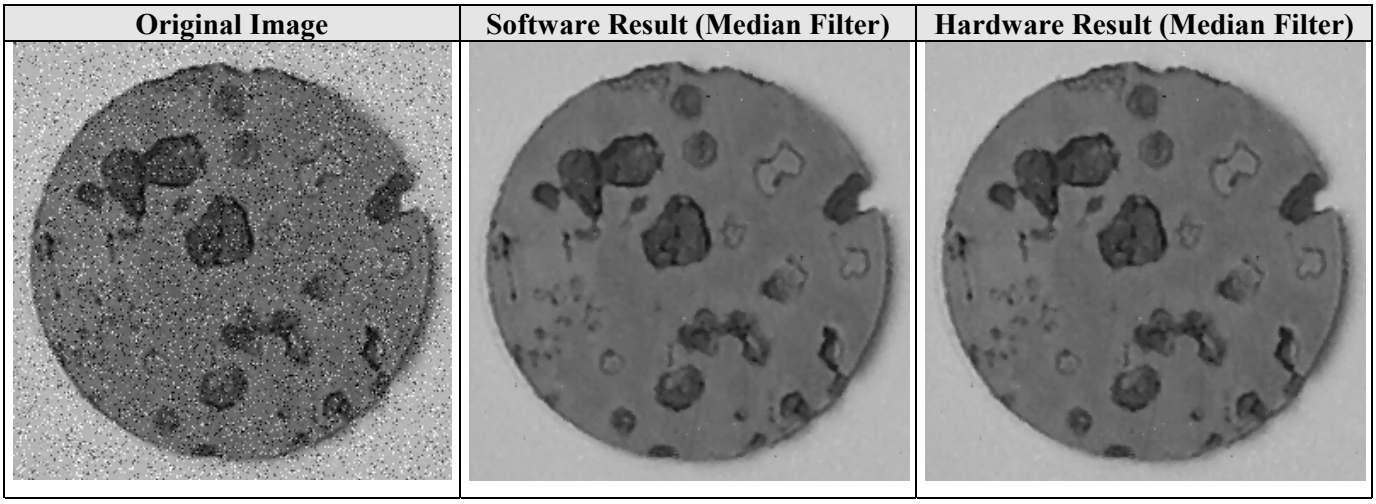

Figure 10: Results obtained by the median filter (software and hardware implementation).

It should be kept in mind that the hardware module has its functional units replicated four times. This fact multiplies the resource use by four. Even more, the module includes the circuitry required by the management of the on-board memory. In spite of everything, the designed hardware module presents an efficient use of the FPGA resources, requiring a number of CLBs less than 28%. This low resource use is useful for future work lines, combining several modules in the FPGA at the same time, so avoiding intermediate reconfiguration times. Pipelines could be designed where each stage is an image processing operation. The images would pass each one of the stages until obtaining the final resulting image.

The fourth row presents the maximum frequency admitted by our hardware module. The minimum period (inverse datum) is also indicated. The hardware module admits a maximum clock frequency lightly higher than 16 MHz. Note that it includes a pixel sorting and selection circuit. This kind of circuits has a lot of logical levels, generating a great quantity of propagation delays (logic and routing delays). These facts increase the minimum clock period, and therefore, decrease the maximum frequency.

We are now studying the use of FPGAs as part of an industrial inspection system to classify cork stoppers according to their quality [10]. We are using image processing techniques to achieve the automatic detection and identification of defects in cork stoppers [11]. In the algorithm we are developing, a median filter is included in order to remove noise from the image and get a better defect detection and an improvement in the later analysis. As an example, Figure 10 illustrates the results obtained when we apply a median filter to a possible input image.

This figure gives us a clearer idea about the utility of median filter. As we can observe, the result obtained by our hardware implementation is identical to the one gotten by the software version, with the advantage of a lower execution time that allows us to meet the real-time requirements of automated visual inspection systems.

6 Conclusions

To configure the HOT2-XL PCI board we are developing a library devoted to image processing. At present, the library is formed by a total of 16 hardware modules, of very diverse types: point, histogram, convolution, mathematical morphology, ... operations. In this work we have shown a study of the architecture and optimizations proposed for the implementation in an FPGA of the median filter.

The practical results illustrate the effectiveness of the proposed architecture and performed optimizations: use of parallelism techniques like replication and pipelining, reduction of the sorting and selection network, optimization of the basic

nodes removing multiplexers, search for a high regularity, reuse of common resources (basic nodes), etc. Everything has allowed us to get real-time processing, a minimum use of resources and a suitable operation frequency.

References

[1] G.A. Baxes. "Digital Image Processing. Principles & Applications", *Wiley & Sons*, (1994).

[2] S. Brown, J. Rose. "FPGA and CLPD Architectures: A Tutorial", *Proc. of IEEE Design & Test of Computers*, **13(2)**, pp. 42-57, (1996).

[3] D.A. Buell, K.L. Pocek. "Custom Computing Machines: An Introduction", *Journal on Supercomput.*, **9**, pp. 219-230, (1995).

[4] R.M. Haralick, L.G. Shapiro. "Computer and Robot Vision", *Addison-Wesley*, **vol. 1**, (1992).

[5] S. Hauck. "The Roles of FPGAs in Reprogrammable Systems", *Proc. of IEEE*, **86(4)**, pp. 615-638, (1998).

[6] B. Morcego, J. Frau, A. Català. "Suavizado de Imágenes en Tiempo Real mediante Filtrado por Mediana Utilizando Arrays Sistólicos", *Proc. of VII DCIS*, pp. 545-546, (1992).

[7] PCI Special Interest Group. "PCI Local Bus Specification - Revision 2.1", http://www.pcisig.com, (1995).

[8] J.L. Smith. "Implementing Median Filters in XC4000E FPGAs", *Xcell*, **23(4)**, pp. 16, (1996).

[9] M.J.S. Smith. "Application-Specific Integrated Circuits", *Addison-Wesley*, (1997).

[10] M.A. Vega, J.M. Sánchez, J.A. Gómez. "Could FPGAs be Used to Classify Cork Stoppers? An Experimental Study", *16th Conference on Design of Circuits and Integrated Systems, DCIS'2001*, pp. 248-253, (2001).

[11] M.A. Vega, J.M. Sánchez, J.A. Gómez. "Cork Stopper Classification using FPGAs and Digital Image Processing Techniques", *EuroMicro Symposium on Digital Systems Design: Architectures, Methods and Tools, DSD'2001*, pp. 270-275, (2001).

[12] J. Villasenor, W.H. Mangione-Smith. "Configurable Computing", *Scientific American*, **276(6)**, pp. 54-59, (1997).

[13] Virtual Computer Corporation. "H.O.T. II Hardware Guide. Version 2.0", (1999).