Novel Hardware Implementation of Adaptive Median Filters

Conference Paper · May 2008

DOI: 10.1109/DDECS.2008.4538766 · Source: IEEE Xplore

2 authors, including:

Zdenek Vasicek
Brno University of Technology
123 PUBLICATIONS **2,568** CITATIONS

SEE PROFILE

Novel Hardware Implementation of Adaptive Median Filters

Zdenek Vasicek
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
E-mail: vasicek@fit.vutbr.cz

Lukas Sekanina
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
E-mail: sekanina@fit.vutbr.cz

***Abstract*—A new FPGA implementation for adaptive median filters is proposed. Adaptive median filters exhibit better filtering properties than standard median filters; however, their implementation cost is higher. Proposed architecture was optimized for throughput allowing 300M pixels to be filtered per second. The best performance/cost ratio exhibits the adaptive median filter which utilizes filtering window 7x7 pixels and can suppress shot noise with intensity up to 60%. In addition to filtering, adaptive median filters can be also used as detectors of corrupted pixels (detection statistics).**

I. Introduction

In past years, *linear filters* became the most popular filters in image signal processing. The reason of their popularity is caused by the existence of robust mathematical models which can be used for their analysis and design. However, there exist many areas in which the *nonlinear filters* provide significantly better results. The advantage of nonlinear filters lies in their ability to preserve edges and suppress the noise without loss of details. The success of nonlinear filters is caused by the fact that image signals as well as existing noise types are usually nonlinear.

Due to the imperfections of image sensors, images are often corrupted by noise. The impulse noise is the most frequently referred type of noise. In most cases, impulse noise is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware, or errors in the data transmission. We distinguish two common types of impulse noise; the *salt-and-pepper noise* (commonly referred to as intensity spikes or speckle) and the *random-valued shot noise*. For images corrupted by salt-and-pepper noise, the noisy pixels can take only the maximum or minimum values. In case of the random-valued shot noise, the noisy pixels have an arbitrary value. It is very difficult to remove this type of noise using linear filters because they tend to smudge resulting images.

Traditionally, the impulse noise is removed by a median filter which is the most popular nonlinear filter. Its hardware implementation is straightforward and does not require many resources. However, the standard median filter gives a poor performance for images corrupted by impulse noise with higher intensity. A simple median utilizing 3×3 or 5×5-pixel window is sufficient only when the noise intensity is less than approx. 10-20%. When the intensity of noise is increasing, a simple median filter remains many shots unfiltered. Thus more advanced techniques have to be utilized. In order to overcome this shortage, various approaches were proposed in the recent years (see a survey of the methods, e.g. in [1]). Among the most known techniques we can include: switching median filters [2], weighted median filters [3], weighted order statistic filters [4] and adaptive median filters [5]. Apart from median filters, some better algorithms exist (e.g.[6]). However, because they do not use the concept of a small filtering window, their hardware implementations do not bring any benefits for a reasonable cost.

Almost all alternatives to median filters have already been implemented in hardware [7], [8], [9], [10], [11], [12]. There is one exception, *adaptive median filter*, whose efficient hardware implementation will be described in this paper. In comparison with other approaches, the adaptive median filter provides significantly better results (in terms of visual quality of filtered images) especially for images corrupted with high noise intensity [5]. The main advantage of the adaptive median filter is that it modifies only corrupted pixels. Standard median filters modify almost all pixels of the image. In addition to filtering, adaptive median filters can be also used as detectors of corrupted pixels (detection statistics) [13], [6].

This paper deals with a highly pipeline hardware realization of the adaptive median filter which is optimized for high throughput. Proposed solution is implemented in an FPGA and its performance is compared against various filters on a set of test images. The goal is to provide a filter suitable for a high performance real-time processing of images corrupted by shot noise of various intensities.

II. Sliding window function

As spatial filters operate with pixel values in the neighborhood of the center pixel (so-called filter or observation window), it is necessary to implement a *local neighborhood function* (sometimes referred to as *a sliding window function*). This function is applied independently on all pixel locations and is typically invariable for all locations (i.e. spatially invariant).

Figure 1 shows the most common hardware architecture of the sliding window function that uses the row buffers. This approach assumes that one image pixel is read from memory in

978-1-4244-2277-7/08/$25.00 ©2008 IEEE

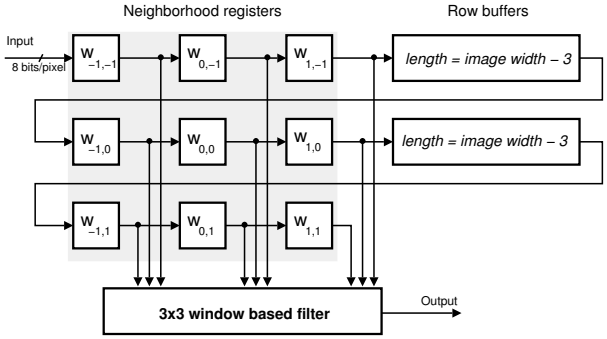


Fig. 1. Implementation of a 3×3 filter window. Row buffers are used to reduce the memory access to one pixel per clock cycle.

one clock cycle. The pixels are read row by row. When buffers are filled (which is done with a fixed latency), this architecture provides the access to the entire pixel neighborhood every clock cycle.

While the architecture places the lowest demand on external memory bandwidth, the highest demand is placed on internal memory bandwidth. Because modern FPGA devices contain large amount of embedded memory, this approach does not cause problems.

The length of the shift registers depends on the width of the input image. In order to implement a sliding window, several image rows have to be stored. The number of rows corresponds to the window size. Another approach is to choose a fixed row length and divide the input image into strips. However, this method leads to decreasing of the performance due to data overlapping (when compared to the usage of full-length row buffers).

If the embedded memory is not available, it is necessary to access the external memory for more than one pixel in one clock cycle. This approach can be efficient only for small window sizes. Hence, it is rarely used in high performance image processing.

As the cost of buffers implementation depends on the size of the input image and as the buffers have to be implemented for every window-based spatial filter, we will not consider this implementation cost in the comparisons which will be performed in Section VI.

III. K-TH ORDER STATISTIC FILTERS

Existing architectures of median filters can be divided into three classes [14]: array-based architectures, stack filter-based architectures, and sorting network-based architectures.

The array architectures use a large number of simple processors arranged into a systolic linear array. Each processor processes one value of the filter window. Even if the processors can be pipelined and can provide high throughput, this architecture is not suitable for manipulating with large windows. Unfortunately, large windows are typical for adaptive median filters.

The most efficient approach is based on stack filters. A stack filter uses a transformation of process of filtration into the binary domain. This transformation uses a threshold decomposition. Processing in the binary domain is very efficient and can be easily parallelized. The main disadvantage of this approach is the requirement for a high number of decomposition levels which depends exponentially on the number of bits used to represent each pixel. On the other hand, in the serial bitwise version, the stack filters usually allow the most area efficient implementation [11].

Sorting networks-based architectures implement the *rank order filters*. The samples of observed filter window are sorted by a sorting network (SN). Then, the value in the middle of sorted sequence represents the median value. As the sorting network can be easily pipelined, the approach provides the best performance. There exist different types of sorting networks. We will use SNs based on the bitonic sort and Batcher's odd-even merge sort. These algorithms differ in the implementation cost.

The most popular filtering techniques developed to suppress impulse noise in images operate on the ordered values within the observation window. This approach is known as *order statistic filtration* and will be introduced in the following paragraphs.

A. Order statistic filters

Consider a sequence $\{x_1, x_2, \ldots, x_N\} = \{x_i\}$, $1 \leq i \leq N$ that consists of N elements generated by a random variable X. Let $\{x_i\}$ be arranged in ascending order so that

$$x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(k)} \leq \ldots \leq x_{(N-1)} \leq x_{(N)}.$$

Then, element $x_{(k)} = S_{(k)}\{x_i\}$ is so-called *k-th order statistic*. Note that element $x_{(1)}$ corresponds to the *minimum* of the observed sequence and $x_{(N)}$ to the *maximum*. In case that $k = (N+1)/2$, where N is odd, $x_{(k)}$ is the *median* of the given sequence.

Let M be the length of the filter window, $M = 2L+1$, and $\{x_i\}$ is the input sequence, $1 \leq i \leq N$ and $N \geq M$. Then the filter defined by specifying its output y_j ($j = L+1, \ldots, N-L$) as

$$y_j = S_{(k)}\{x_{j-L}, \ldots, x_{j+L}\}$$

is denoted as the *k-th order statistic filter* (OSF). It is obvious that if the $k = (N+1)/2$ then the k-th order statistic filter defines the standard median filter.

So-called weighted OSF [14] assigns a weight to every element of the observation window. This generalization allows the usage of some elements of window more than once. On contrary, some elements need not to be included into the process of filtration.

As each pixel of a given image can be treated as a random variable, statistic order filter can be used for the filtration of the images. However, in this case we need a two dimensional variant of statistic filter which can be obtained as an extension of the one-dimensional case mentioned above. Instead of one-dimensional observed sequence $\{x_i\}$, we have to consider a two-dimensional matrix $[x_{i,j}]$. Each element of this matrix corresponds to one pixel of observed input image. Similarly,

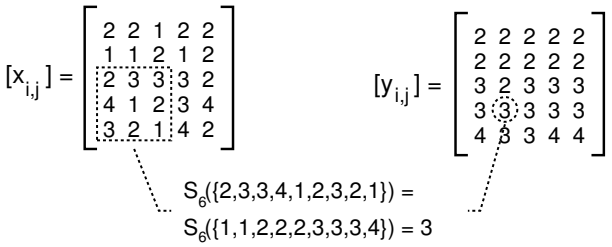


Fig. 2. Filtration using a two dimensional 6th order statistic filter (a 3×3 filter window is used)

the output of the filter is a two-dimensional matrix $[y_{i,j}]$ (see Figure 2). Note that the two-dimensional statistic order filter does not have to use every element of the rectangular filtering window.

B. Sorting networks

The concept of sorting networks was introduced by Knuth [15]. Sorting network is defined as a network of elementary operations denoted as *compare&swap* elements (sometimes called comparators) that sorts all input sequences. A compare&swap (CS) of two elements (a, b) compares a and b and exchanges (if it is necessary) the elements in order to obtain sorted sequence. A sequence of compare&swap operations depends only on the number of elements to be sorted, not on the values of the elements. The main advantage of the sorting network is that the sequence of comparisons is fixed. Thus it is suitable for parallel processing and pipelined hardware implementation. In hardware, CS is implemented using two multiplexers that are controlled by means of a comparator that determines the maximum of the two.

The sorting network can be constructed using a sorting algorithm which must be data independent (i.e. the sequence of CS components must not depend on the input values). The number of CS components and latency are two crucial parameters of any sorting network. By latency we mean the minimum number of groups of compare&swap components that must be executed sequentially. Chosen sorting algorithm influences the number of required CS components. Hence, we used two sorting algorithms which are able to provide low number of CS components: bitonic sorter and Batcher's odd-even merge sorter. When used to implement median circuits, these algorithms provide solutions with the same latency. However, these solutions differ in the number of CS components and registers. Note that some additional registers have to be included into the design in order to allow the pipeline processing.

The bitonic sorter [16] is developed on the basis of the 0-1 principle [15]. It is based on merging of two so-called bitonic sequences. A 0-1-sequence is called bitonic if it contains at most two changes between 0 and 1. The main idea is to recursively divide the input sequence into several parts. In each part, bitonic sequences are created and subsequently merged in order to create 1) another larger bitonic sequence and 2) sorted sequence. After all merging tasks, the sequence is sorted. Structure of the 9-input sorting network created using the bitonic sorter is depicted in Figure 3.

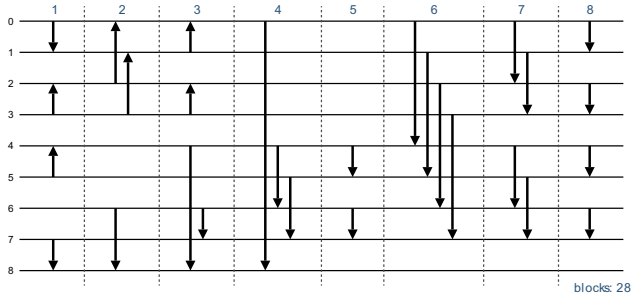


Fig. 3. Structure of 9-input sorting network (bitonic sorting network). Each vertical line represents one compare&swap operation. The arrow determines the position of maximum of the two inputs.

Similarly, the Batcher's odd-even merge sort [16] algorithm is based on sorting of two halves of the input sequence. The sorting is followed by merging of two shorter sequences into the larger sorted sequence. The input sequence is recursively spitted into sequences of odd and even elements.

Table I contains the number of compare&swap components, the number of registers (REG) and the latency (delay) of the median filters obtained by various approaches. The implementation cost of the optimal median circuits is also included. However, the optimal median implementation is known only for some problem instances. As we can see, the odd-even merge sort produces more efficient implementations in comparison with the bitonic sorter.

TABLE I
COST OF HARDWARE IMPLEMENTATION OF VARIOUS MEDIAN CIRCUITS

#	optimal			bitonic SN			oemerge SN		
inputs	CS	REG	delay	CS	REG	delay	CS	REG	delay
3	3	2	3	3	2	3	3	2	3
5	7	4	5	9	5	5	8	5	5
7	13	10	7	18	4	6	14	5	6
9	19	15	9	28	12	8	22	14	8
25	99	121	24	153	58	15	113	70	15
49	-	-	-	427	155	21	313	203	21
81	-	-	-	876	439	28	661	541	28

It is obvious that we can implement standard as well as weighted OSF of an arbitrary order using of an appropriate sorting network. In fact, particular SN computes all orders of OSF in parallel. This property will be utilized in proposed adaptive median filter implementation.

IV. ADAPTIVE MEDIAN FILTER

The adaptive median filter (AMF) can be defined in several ways [17], [5]. We will use the definition based on the order statistic. In this sense, AMF can be considered as *iterative order statistic filter*. The iterative processing was introduced in order to detect and replace *corrupted* pixels only. In each iteration, filtering windows of *different* sizes are utilized.

In order to simplify the description, we will deal only with one filter window located at position (u, v). Let a two-dimensional matrix $[x_{i,j}]$ describe the input image and W is

the size of the filtered window. Let the sequence $[w_{k,l}]$ be the output of a local neighborhood function which contains just $N = W \times W$ samples of filter window located at position (u, v) (assume that W is odd). Let x_{uv} denote the value of pixel $x_{u,v}$ which corresponds to the value of a pixel at position (u, v) of the input image. Let y_{uv} be the output of the AMF located at position (u, v). The algorithm of AMF is as follows:

Step 1 Initialization

Start with the smallest windows size $W = 3$. Let the maximum window size be W_{max} (again, an odd number).

Step 2 Computation of order statistic

Let $x_{min} = S_{(0)}([w_{k,l}])$ be the output of the 0-th order statistic filter. $x_{max} = S_{(N)}([w_{k,l}])$ is the output of the N-th order statistic filter and $x_{med} = S_{((N+1)/2)}([w_{k,l}])$ is the output of the median filter.

Step 3 Evaluation of the terminating condition

If the condition $x_{min} < x_{med} < x_{max}$ is satisfied then the processing ends with the computation of the output value which is defined as follows: If $x_{min} < x_{uv} < x_{max}$ then the pixel is not corrupted by noise and the output value is the value of the original pixel, i.e. $y_{uv} = x_{uv}$. If $x_{min} < x_{uv} < x_{max}$ is not satisfied then the output value is the median of the window, i.e. $y_{uv} = x_{med}$. If the condition is not satisfied then the computation continues.

Step 4 Increasing of the window size

If the condition $x_{min} < x_{med} < x_{max}$ is not satisfies, it can be interpreted as follows. If many pixels have the same value then it is impossible to determine (with the current window size) whether the pixels are corrupted with high intensity noise or whether it is the constant area with all pixels of the same color. This is the reason why the window size has to be increased.

If the window W is smaller than W_{max}, increase the size of the window, i.e. $W = W + 2$, and repeat the computation from step 2. If the size of the window W reaches the maximum value W_{max}, the processing ends and the output value is defined as $y_{uv} = x_{med}$.

V. Proposed architecture

A. Non-recursive adaptive median filter

Although the adaptive median filter is defined as an iterative filter, the result can be computed in a two-step process. The idea is to implement a set of sorting networks of different number of inputs (from 3×3 to $W_{max} \times W_{max}$). The minimum, maximum and median value of each sorting network is utilized. As these sorting networks have different latencies it is necessary to include registers at suitable positions to synchronize the computation. In the second step, the outputs of sorting networks are combined together using a simple combination logic (Figure 5).

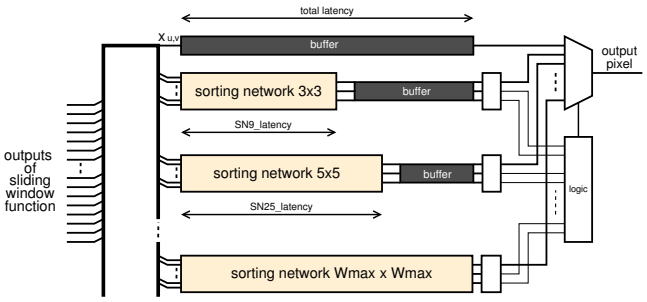


Fig. 5. Hardware implementation of the adaptive median filter

B. Recursive adaptive median filter

The implementation cost can be reduced if only one $W_{max} \times W_{max}$-input sorting network is used. This sorting network is used iteratively for all filtering windows. It is necessary to ensure that unused inputs are initialized to correctly calculate the minimum, maximum and median value for each filtering window. Immediate results have to be stored to registers. However, the throughput of this solution is three times lower (in case that $W_{max} = 7$) in comparison to a non-recursive implementation.

VI. Results of synthesis

The median filters as well as adaptive median filters were described in VHDL, simulated using ModelSim and synthesized using Xilinx ISE tools to Virtex II Pro XC2vp50-7 FPGA. The implementation costs is expressed in terms of slices. Our FPGA contains 23616 slices in total.

Table II provides the results of synthesis of median circuits for various windows sizes and various architectures. The results corresponds with the costs given in Table I. The odd-even merge sort-based implementations require fewer slices in comparison with the bitonic sorter-based implementation.

Median filters are implemented as pipeline circuits with the maximal degree of parallelism. The solutions can achieve the throughput of 300M calculated medians (i.e. processed filter windows) per second. When we consider an image containing 1024×1024 pixels, the proposed architecture is able to filter approx. 280 images per second.

TABLE II
Results of synthesis of common median filters

# inputs	Number of slices			max. freq.
	optimal	bitonic SN	oe-merge SN	
9 (3x3)	268	297	289	305 MHz
25 (5x5)	1506	1706	1582	305 MHz
49 (7x7)	unknown	4815	4426	303 MHz
81 (9x9)	unknown	10315	9719	302 MHz

Results of synthesis of the proposed adaptive median filter are summarized in Table III. Adaptive median filter with filtering window 7x7 exhibits a very good performance/cost ratio in comparison to standard median filters. This filter occupies approx. 30% of the chip and is able to remove noise up to 60% intensity. As the design of AMF is pipelined

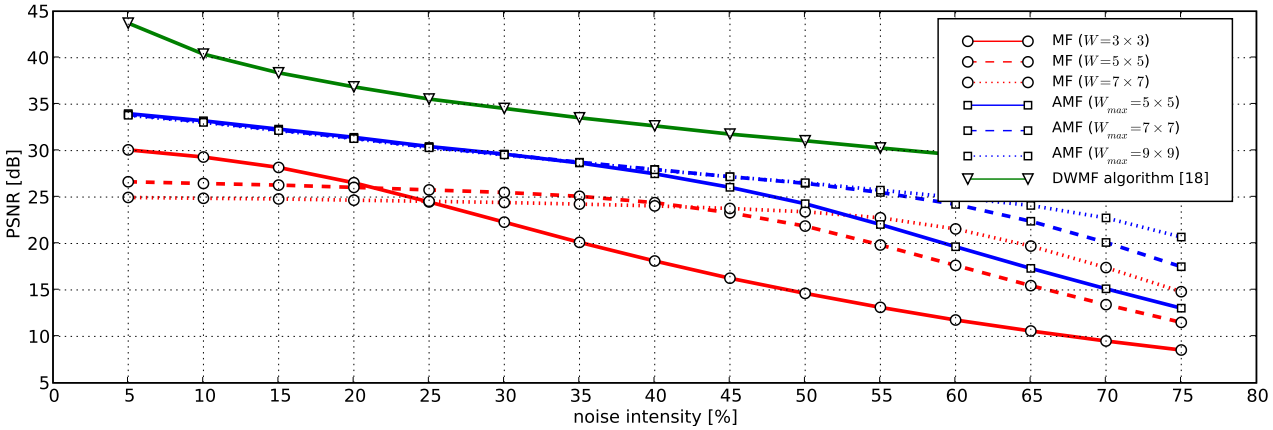


Fig. 4. Comparison of various image filters using a set of 25 test images corrupted by salt-and-pepper noise of intensity 5-75%.

and without iterations, it provides the same performance as standard median filters (i.e. 300M processed pixels per second).

TABLE III
RESULTS OF SYNTHESIS OF PROPOSED ADAPTIVE MEDIAN FILTER

W_{max}	SN bitonic		SN oe-merge		Latency
	# slices	max. freq	# slices	max. freq	[delay]
5x5	2220	305 MHz	2024	303 MHz	15
7x7	7297	302 MHz	6567	298 MHz	21
9x9	18120	302 MHz	16395	298 MHz	28

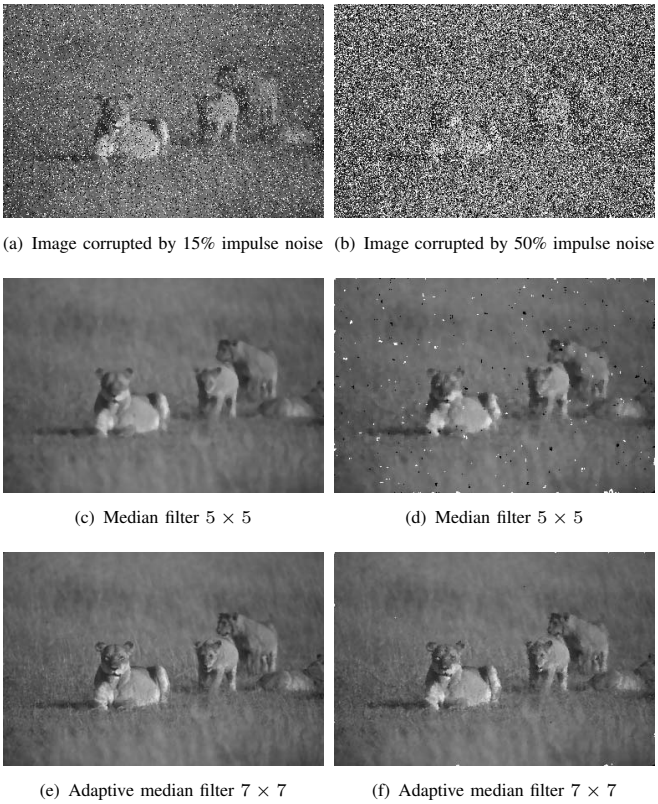
VII. Evaluation of Filtering Properties

Filtering properties of adaptive median filters (with filtering windows 5x5, 7x7 and 9x9) and standard median filters (with filtering windows 3x3, 5x5 and 7x7) were compared on 25 test images. All images were corrupted by salt-and-pepper noise of intensity 5-75%. The results were also compared the best known software solutions [18] which utilizes filtering windows of unlimited size. Figure 4 summarizes obtained results. The visual quality of filtered images is numerically expressed by the peak signal-to-noise ratio (PSNR) which is calculated as

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{MN} \sum_{i,j} (v[i,j] - w[i,j])^2}$$

where $N \times M$ is the size of image, v denotes the filtered image and w denotes the original image.

Increasing the size of filtering window allows the standard median filter to improve the PSNR; however, this approach fails when the noise intensity is higher that approx. 10-20%. Because the standard median filters modify almost all pixels, images become smudged and detail less. Adaptive median filters work correctly also for lower noise intensities because they try to use the smallest possible window and so modify only corrupted pixels. The size of filtering window influences the quality of filtering when noise is of more than 40% intensity. Figure 6 give examples of images filtered using different filters.



(a) Image corrupted by 15% impulse noise (b) Image corrupted by 50% impulse noise

(c) Median filter 5 × 5 (d) Median filter 5 × 5

(e) Adaptive median filter 7 × 7 (f) Adaptive median filter 7 × 7

Fig. 6. Filtration of images corrupted by impulse noise with a low intensity (left column) and a high intensity (right column).

VIII. Conclusions

In this paper, a new FPGA implementation for adaptive median filters was proposed. Adaptive median filters exhibit better filtering properties than standard median filters; however, their implementation cost is higher. Proposed architecture was optimized for throughput allowing 300M pixels to be filtered per second. The best performance/cost ratio exhibits the adaptive median filter which utilizes filtering window 7x7 pixels and can suppress shot noise with intensity up to 60%.

ACKNOWLEDGEMENTS

This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 – *Security-Oriented Research in Information Technology*.

REFERENCES

[1] S. Schulte, M. Nachtegael, V. D. Witte, D. V. der Weken, and E. E. Kerre, "Fuzzy impulse noise reduction methods for color images," in *Computational Intelligence, Theory and Applications International Conference 9th Fuzzy Days in Dortmund*. Springer Verlag, 2006, pp. 711–720.

[2] W. Zhou and Z. David, "Progressive switching median filter for the removal of impulse noise from highly corrupted images," *IEEE Trans On Circuits and Systems: Analog and Digital Signal Processing*, vol. 46, no. 1, pp. 78–80, 1999.

[3] D. R. K. Brownrigg, "The weighted median filter," *Commun. ACM*, vol. 27, no. 8, pp. 807–818, 1984.

[4] S. Marshall, "New direct design method for weighted order statistic filters," *VISP*, vol. 151, no. 1, pp. 1–8, February 2004.

[5] H. Hwang and R. Haddad, "Adaptive median filters: new algorithms and results," *IP*, vol. 4, no. 4, pp. 499–502, April 1995.

[6] M. Nikolova, "A variational approach to remove outliers and impulse noise," *J. Math. Imaging Vis.*, vol. 20, no. 1-2, pp. 99–120, 2004.

[7] S. A. Fahmy, P. Y. K. Cheung, and W. Luk, "Novel fpga-based implementation of median and weighted median filters for image processing," in *FPL*, T. Rissa, S. J. E. Wilton, and P. H. W. Leong, Eds. IEEE, 2005, pp. 142–147.

[8] D. Caban, "FPGA implementation of positional filters," in *Design of Embedded Control Systems*. Springer-Verlag, 2005, pp. 243–249.

[9] R. Maheshwari, S. S. S. P. Rao, and E. G. Poonach, "FPGA implementation of median filter," in *VLSI Design*. IEEE Computer Society, 1997, pp. 523–524.

[10] C. Chakrabarti, "Sorting network based architectures for median filters," *Transaction on Signal Processing*, 1994.

[11] P. Lakamsani, R. Yang, B. Zeng, and M. Liou, "Design and implementation of a programmable stack filter," in *ICIP94*, 1994, pp. 664–667.

[12] C. Chakrabarti, "Novel sorting network-based architectures for rank order filters," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 2, no. 4, pp. 502–507, 1994.

[13] S.-Q. Yuan and Y.-H. Tan, "Erratum to "impulse noise removal by a global-local noise detector and adaptive median filter": [signal processing 86 (8) (2006) 2123-2128]," *Signal Processing*, vol. 87, no. 5, p. 1171, 2007.

[14] C. Chakrabarti and L. E. Lucke, "VLSI architectures for weighted order statistic (WOS) filters," *Signal Processing archive*, vol. 80, no. 8, pp. 1419–1433, 2000.

[15] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[16] K. E. Batcher, "Sorting networks and their applications," in *AFIPS Spring Joint Computing Conference*, 1968, pp. 307–314.

[17] H. Hwang and R. A. Haddad, "New algorithms for adaptive median filters," in *Proc. SPIE Vol. 1606, Visual Communications and Image Processing '91*, Nov. 1991, pp. 400–407.

[18] Y. Dong and S. Xu, "A new directional weighted median filter for removal of random-valued impulse noise," *Signal Processing Letters*, vol. 14, no. 3, pp. 193–196, 2007.