J Real-Time Image Proc (2019) 16:2015–2026
DOI 10.1007/s11554-017-0705-4

ORIGINAL RESEARCH PAPER

FPGA implementation of an adaptive window size image impulse noise suppression system

Parham Taghinia Jelodari[1] · Mojtaba Parsa Kordasiabi[1] · Samad Sheikhaei[1] · Behjat Forouzandeh[1]

Received: 17 December 2016 / Accepted: 4 July 2017 / Published online: 20 July 2017

Abstract The conventional method for image impulse noise suppression is standard median filter utilization, which is satisfying for low noise densities, but not for medium to high noise densities. Adding a noise detection step, as proposed in the literature, makes this algorithm suitable for higher noises, but may degrade the performance at low noise densities. An adaptive switching median-based (ASM) algorithm has been used in this paper for noise suppression. First, the algorithm is modified to achieve a higher PSNR, especially for low noise densities. Then, the structure of the modified algorithm is improved to obtain higher operating speed in hardware implementation, for real-time applications. The implemented algorithm works in two steps, detection and filtering. The noise detection method is enhanced, by merging the amount of memory used for the algorithm implementation. As a result, less hardware resources are required, while the chance of false noise detection is reduced, due to the improvement made in the algorithm. In the filtering step, an adaptive window size is used, based on the measured noise density. This improved algorithm is adopted for more efficient hardware implementation. In addition, high parallelism is utilized to boost the operating frequency, and meanwhile, clock gating is used to lower power consumption. This architecture, then, has been implemented physically on an FPGA, and an operating frequency of 93 MHz is achieved. The hardware requirement is approximately 10,000 4-input LUTs, and the processing time for a 512 $\times$ 512 pixels image is measured at 12 ms.

Keywords Impulse noise · Noise suppression · Adaptive switching median-based algorithm · ASM algorithm · Adaptive window size · FPGA implementation · DE2-115 board

✉ Samad Sheikhaei
sheikhaei@ut.ac.ir

Parham Taghinia Jelodari
p.taghinia@ut.ac.ir

Mojtaba Parsa Kordasiabi
m.parsa@ut.ac.ir

Behjat Forouzandeh
bforooz@ut.ac.ir

1 School of Electrical and Computer Engineering, University of Tehran, Tehran, Iran

1 Introduction

Image restoration is one of the most important fields in digital image processing, and its aim is to restore the corrupted image from the degraded one [1–4]. Denoising is an image restoration technique, and many researches and studies have been done with the purpose of introducing more efficient algorithms. An algorithm would be considered more efficient, which suppresses the noise as much as possible, while resulting in a less blurred image. Noise suppression as a low-level processing has a direct effect on higher-level processes, such as image segmentation and recognition [5–9]. In other words, less efficient algorithms for noise suppression lead to more faults in higher-level image processing. With developments in modern image capturing devices technology, noise sensitivity rises that leads to noisier images [10]. Hence, there is an ever growing demand for more sophisticated and efficient algorithms.

Noise suppression algorithms are divided into three major groups: noise suppression without detection, noise suppression with detection and combined method. The first group has low complexity, simple implementation and is appropriate only for low noise densities, while the second and third groups are more complex, more efficient and able to suppress higher noise densities.

Among the first group, the center-weighted median (CWM) filter [11] is outstanding. In this method, the center pixel of the neighborhood is given a constant even number weight, in order to increase the effect of the center pixel. Output of the CWM with a $L \times L$ window equals the output of a standard median filter, in which the center pixel has weight of 2 N, but the neighborhood pixels have weight of 1. In this way, the effective size of the filter equals "$2\,N + (L \times L) - 1$." This method has excellent performance in low noise densities, owing to more emphasis on the center pixel. However, by increasing the noise density, the performance decreases notably, because it tends to substitute the corrupted value or near-corrupted value for the center pixel. In fact, in higher noise densities, standard median filter performs better than the CWM filter. The CWM method also has a simple hardware implementation.

Among the second group, progressive switching median (PSM) filter [12] is worth mentioning here. In this method, the filtering and detection are performed iteratively, in order to boost the performance. This method has proper performance for noise densities between 10 and 70%. The problem with the hardware implementation of this method is that it needs creation of some flag images and grayscale images in either steps of detection and filtering. Thus, the processing time and the memory usage are considerably high, which suits this method only for PC simulations and not for real-time systems.

Another outstanding work in the second group is the *peak-and-valley* filter [13]. This filtering method is a nonlinear recursive filter which is neither a statistical filter nor a morphological filter. The 1D version of this method is used for impulsive noise suppression. This method removes all the peaks and valleys of the neighborhood. It is based on two steps of cutting filling and filling cutting, and this act is iterated recursively to the point that all peaks and valleys of the image are removed. This algorithm suffers from deficient noise detection, and also because of the excessive smoothing of the image, it blurs the image significantly.

There are some worthy detection algorithms in the literature, from which we can mention the Laplace-based detection algorithm that exploits four 1D directional cores, which are sensitive to edges in four directions. This method also suffers from high computational complexity, which results in high processing time and resource usage.

Finally, among the last group, *tri-state median filter* is highlighted [14]. This method consists of three filters of standard median, identity and center-weighted filters and also a switching logic. The disadvantage of this method is its tri-state switching logic that relies on a pre-defined threshold. It is obvious that the threshold value varies from one image to another, and using a pre-defined fixed threshold offers improper choice between the filters, and eventually lowers the performance for certain images.

Among the novel efficient algorithms, adaptive switching median-based (ASM) filter for impulse noise removal seems to be an appropriate method for low-to-medium noise densities suppression. This method has both high efficiency and moderate complexity that makes it more desirable for real-time hardware implementation [15–18].

In this paper, a novel system is introduced for low-to-medium noise densities suppression, based on a modified ASM algorithm. The system is made up of two sections: noise detection and noise filtering. A 512×512 8-bit resolution grayscale image has been chosen as a test image. This impulsive noise suppression system has been implemented physically on an FPGA of Altera Cyclone IV family.

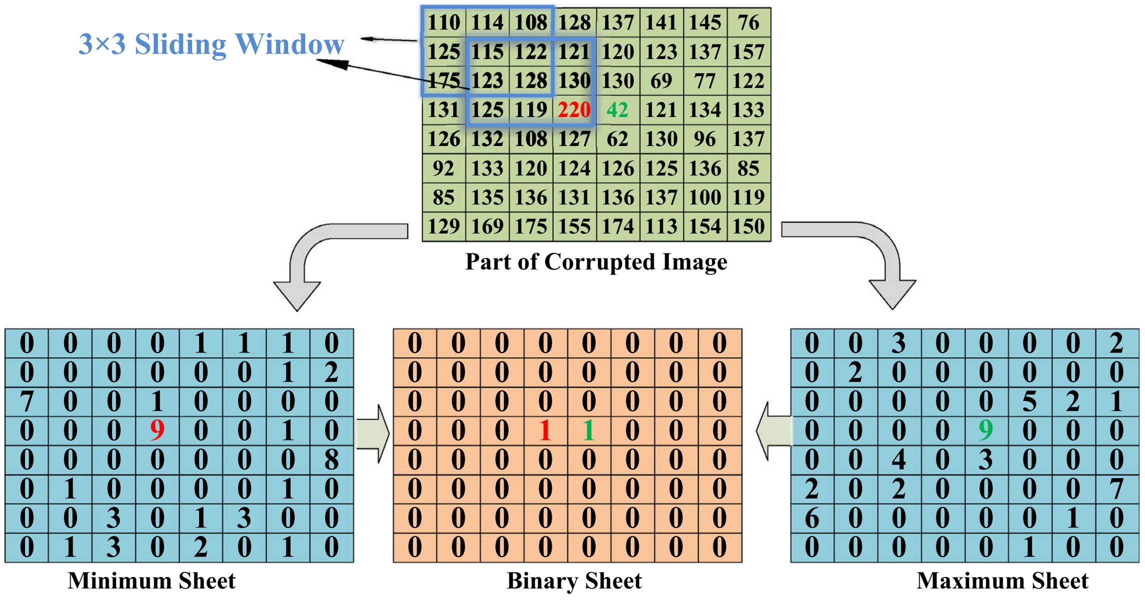
We claim novelty for the proposed improvement on the ASM algorithm, which enhances the PSNR performance, especially for low noise densities. Also, the structure is adopted for high parallelism and speed improvement in the hardware for real-time operation.

This paper focuses on three major subjects, including the algorithm, the architecture and the physical implementation. In Sect. 2, the modified ASM algorithm is discussed. In Sect. 3, novel architectures are proposed for noise detection and noise filtering systems. In Sect. 4, physical implementation of the noise suppression system on a DE2-115 evaluation board is illustrated. Sect. 5 provides a discussion on the system implementation, and finally, in Sect. 6, the paper is concluded.

2 Algorithm

2.1 The primary ASM algorithm

An ASM filter is chosen, due to its high efficiency and simplicity. This suppression method that belongs to the second major group mentioned above has two steps. In the first step, noise detection mechanism detects the corrupted pixels, and in the second step, only the corrupted pixels are filtered. Therefore, the method is considered as a two-pass type filter. A 3×3 window sweeps the whole image from top left to bottom right as shown in Fig. 1. In each window, maximums and minimums are detected. If a pixel is maximum in the window, the corresponding address in the maximum map sheet is incremented, and if a pixel is minimum, the corresponding address in the minimum map sheet is incremented. Because it is a 3×3 size window,



Part of Corrupted Image

110	114	108	128	137	141	145	76
125	115	122	121	120	123	137	157
175	123	128	130	130	69	77	122
131	125	119	220	42	121	134	133
126	132	108	127	62	130	96	137
92	133	120	124	126	125	136	85
85	135	136	131	136	137	100	119
129	169	175	155	174	113	154	150

Minimum Sheet

0	0	0	0	1	1	1	0
0	0	0	0	0	0	1	2
7	0	0	1	0	0	0	0
0	0	0	9	0	0	1	0
0	0	0	0	0	0	0	8
0	1	0	0	0	0	1	0
0	0	3	0	1	3	0	0
0	1	3	0	2	0	1	0

Binary Sheet

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Maximum Sheet

0	0	3	0	0	0	0	2
0	2	0	0	0	0	0	0
0	0	0	0	0	5	2	1
0	0	0	0	9	0	0	0
0	0	4	0	3	0	0	0
2	0	2	0	0	0	0	7
6	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0

Fig. 1 Procedure of mapping

each pixel appears in nine different windows as the 3 × 3 window moves through the image. At the end of the process, i.e., when the 3 × 3 sliding window reaches the end of the image, the pixels that have value of "9" in each sheet are known as corrupted pixels, because they have been recognized maximum or minimum, nine times in nine different 3 × 3 windows. At the end of the process, the pixels that have value of "9" in each sheet are known as corrupted pixels. At last, the maximum and minimum map sheets are mapped to a binary decision map. In the binary sheet, pixels with value of "9" are mapped to "1" and the rest are mapped to "0" [15]. This procedure is shown for a small part of a test image in Fig. 1.

This method is examined for three test images in Fig. 2 and is proven not to be efficient for low noise densities. Figure 3 indicates the number of noisy-pixel over-estimations versus the real noisy pixels in percentage, for the three test images. As it is obvious from the diagram, in low noise densities, number of healthy pixels that are detected as corrupted pixels rises rapidly. The reason is that in low noise densities, in some regions, there are no corrupted pixels, while the smallest difference between a healthy pixel and its neighborhood pixels is detected as noise. In addition, background pixels that have the same intensity are mapped to both minimum and maximum map sheets and, therefore, are identified as noisy pixels. Hence, the algorithm is required to be modified.

2.2 The proposed modified ASM algorithm

The problem of over-estimation originates from the fact that the two minimum and maximum sheets are not related.

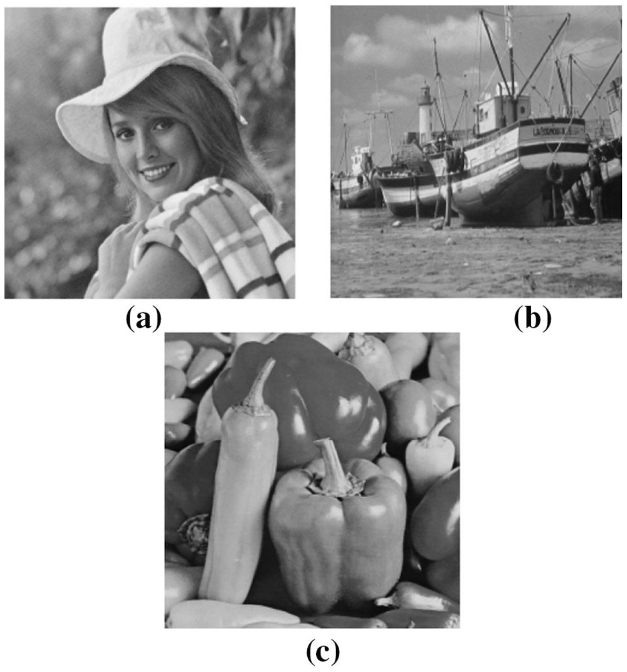


Fig. 2 Three standard test images, **a** Girl, **b** Boat, **c** Pepper

Therefore, the solution is to relate the two sheets to each other. In addition, there is no need for three mapping sheets; one sheet is sufficient for the mapping procedure. Therefore, in the improved algorithm, only one sheet is used for mapping. If a pixel is found to be minimum in a window, the corresponding address in the mapping sheet is decremented and if the pixel is found to be maximum, the corresponding address in the mapping sheet is incremented.

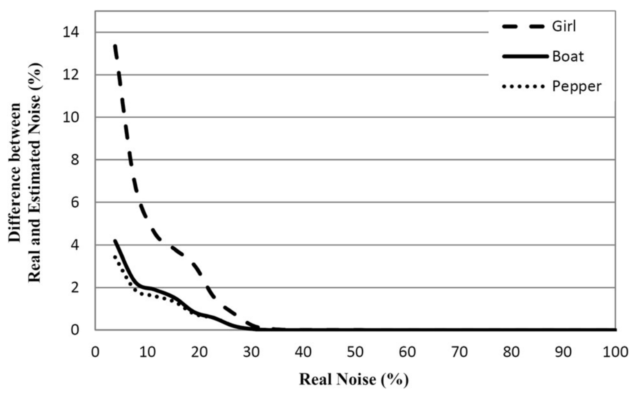


Fig. 3 Difference between real and estimated noise versus real noise in percentage for three images of Fig. 2

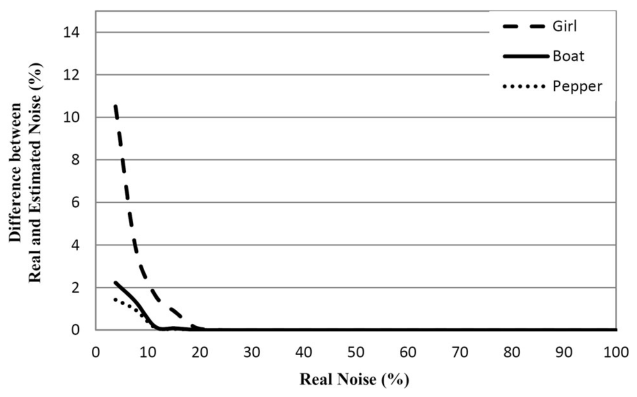
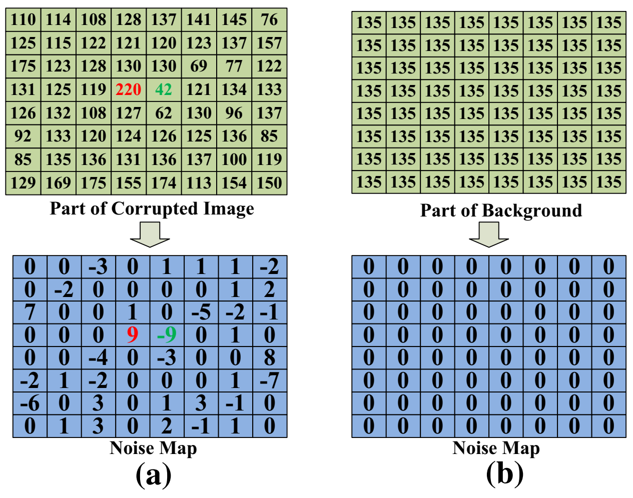


Fig. 5 Difference between real and estimated noise versus real noise in percentage, for three images of Fig. 2, in the proposed modified procedure

Consequently, pixels that have value of “9” are detected as salt noisy pixels, and the ones that have value of “−9” are detected as pepper noisy pixels. Reducing three mapping sheets to one sheet results in lower memory requirements. Figure 4a, b shows the modified procedure of mapping for a typical content part and background part of a test image, respectively, which indicates the effectiveness of the solution for low noise densities. As seen in this figure, no noisy pixel is detected in the background. To investigate the effect of this solution on real images, the test images of Fig. 2 are used. The results shown in Fig. 5 indicate the significant decline of noise over-estimation percentage for the three test images. For example, for 10% noise density in the Boat and Girl images, the number of noisy-pixel over-estimations reduces from 1.98 and 4.57% for the primary algorithm to 0.76 and 1.86% for the modified algorithm, respectively.



Part of Corrupted Image

110	114	108	128	137	141	145	76
125	115	122	121	120	123	137	157
175	123	128	130	130	69	77	122
131	125	119	220	42	121	134	133
126	132	108	127	62	130	96	137
92	133	120	124	126	125	136	85
85	135	136	131	136	137	100	119
129	169	175	155	174	113	154	150

Noise Map

0	0	-3	0	1	1	1	-2
0	-2	0	0	0	0	1	2
7	0	0	1	0	-5	-2	-1
0	0	0	9	-9	0	1	0
0	0	-4	0	-3	0	0	8
-2	1	-2	0	0	0	1	-7
-6	0	3	0	1	3	-1	0
0	1	3	0	2	-1	1	0

(a)

Part of Background

135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135
135	135	135	135	135	135	135	135

Noise Map

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b)

Fig. 4 Modified procedure of mapping for **a** a typical content part and **b** a typical background part of a test image

3 Architecture

Noise Suppression System, itself, consists of *Noise Detection System* and *Noise Filtering System*. *Noise Detection System* initially finds the corrupted pixels, and then, *Noise Filtering System* modifies them. In order to achieve a more precise value for median filtering, a 3 × 3 window is selected for low noise densities, and a 5 × 5 one is used for medium noise densities, in the filtering system. Choosing between the above two choices is performed after the detection step, based on the percentage of noisy pixels detected, and a pre-defined threshold value. For example, for the ranges of 0–30 and 30–60%, window sizes of 3 × 3 and 5 × 5 are selected, respectively.

This decision is made, because in medium noise densities, several pixels in the 3 × 3 window are corrupted; hence, window size is required to be increased in order to achieve larger statistical population.

An 8-bit 512 × 512 standard image of “Boat” has been picked up for evaluating the algorithm. The original test image has been depicted in Fig. 6. In order to show the performance of the proposed method for filtering step, i.e., changing the size of sliding window, different percentages of pixel corruption are applied to the original image, and then, a 3 × 3 detection window is used for detecting noisy pixels. After that, different sizes of filtering windows, i.e., 3 × 3 and 5 × 5, are utilized, and PSNR of the restored images is demonstrated in Fig. 7. As shown in this figure, increasing the size of window, in higher densities of noise, results in PSNR improvement.

For implementation, a RAM memory with 262,144 bytes is needed for storage of both corrupted and processed image in the same place. This RAM is called *RAM1*. Another RAM is required for storage of the noise mapping sheet that is called *RAM2*.

Figure 8 shows the two main blocks of the system implementation, which are *Noise Detection System* and

Fig. 6 The 8-bit 512 × 512 standard image of "Boat"

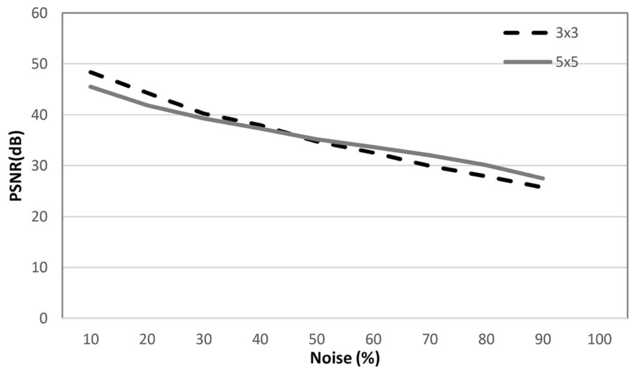


Fig. 7 PSNR comparison for utilizing 3 × 3 and 5 × 5 filtering windows

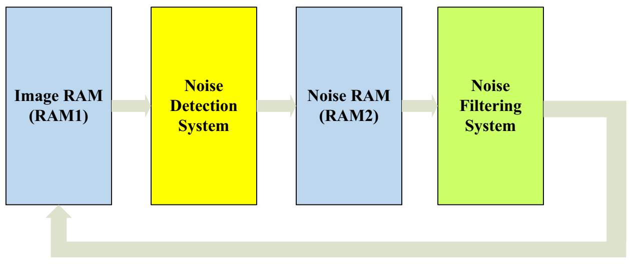


Fig. 8 Main blocks of *Noise Suppression System*

Noise Filtering System. The *Noise Detection System* reads the noisy image from *RAM1* and detects the noisy pixels and stores a map of those noisy pixels in *RAM2*. Then, the *Noise Filtering System* reads both the original image from *RAM1* and noisy pixels' map from *RAM2* and then modifies the corrupted pixels based on the described algorithm and then re-writes the modified pixels back to *RAM1*. Therefore, the original image is replaced with the denoised image, after the algorithm is applied. Since the two blocks do not work simultaneously, there would be no conflict in accessing the memory.

Details of the two main blocks of the system are described in the following.

3.1 Noise Detection System

Noise Detection System consists of major components such as *Window System*, *Window Pixels Addressing*, *Noise Address Detector*, *Controller* and *Parallel/Serial Blocks*. Noise detection uses a 3 × 3 window to sweep the image.

3.1.1 Window System

Window System consists of *Sliding Window* and *Window Controller*, as shown in Fig. 9. The *Sliding Window* is one of the most fundamental components for implementation of space domain filters. It is also called serpentine memory, because of its serpentine movement through the image, as shown in Fig. 10.

The *Sliding Window* indeed converts the 2D signal to 1D, for the following processing. It consists of shift registers and FIFO buffers [19]. The buffer length is equal to the image width minus 3. For implementation of a $(2N + 1)\times(2N + 1)$ window, 2 N FIFO buffers and $2(2N + 1) + 1$ shift registers are required. Consequently, a 3 × 3 window consists of 2 FIFO buffers and 7 shift registers, as shown in Fig. 11. In this figure, W stands for the image width. *Sliding Window* has a pipeline architecture, so that after a definite delay, it outputs 9 pixels per clock cycle.

For automation of the *Sliding Window* operation, a controller called *Window Controller* that handshakes with the *Sliding Window* block is needed. The *Window Controller* reads image pixels located in image RAM, i.e., *RAM1,* from top-left to bottom-right corner of image that is fed serially to the *Sliding Window*. The write and read operations of the buffers are also controlled by this unit.

After the window is formed, it is necessary to map all addresses of the pixels within the window on the basis of a reference address. The reference address is defined as address of a pixel that first window is formed, when it reaches the sliding window. Thereafter, with each clock cycle, the reference address is incremented by one. If the

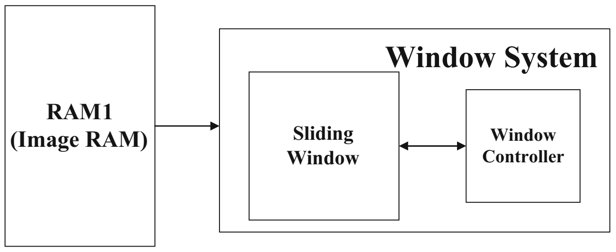


Fig. 9 Structure of the *Window System*

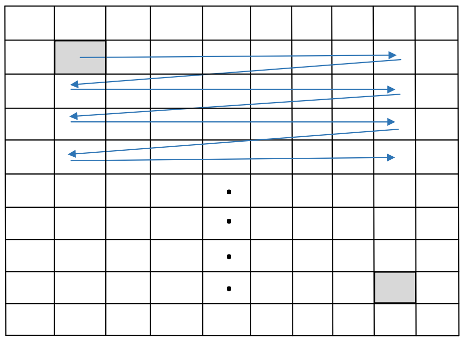


Fig. 10 Movement of window through the image

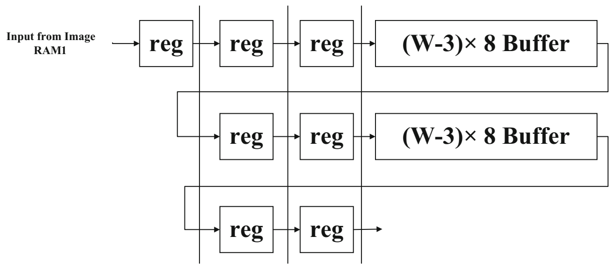
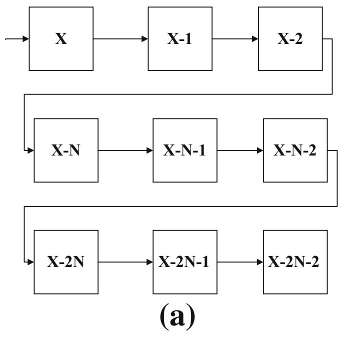


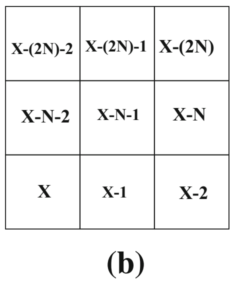
Fig. 11 *Sliding Window* components (*W* is the image width)

reference address is named *X*, addresses of the neighborhood pixels are given according to Fig. 12, where *N* is the image width. Figure 12a shows the procedure of window formation, while Fig. 12b is its equivalent window located on image.

3.1.2 Noise Address Detector

Noise Address Detector is a component of the *Noise Detection System*. This unit is comprised of *Max–Min Filter* and *Max–Min Address Finder*. The *Max–Min Filter* finds the maximum and minimum values within the window. It consists of 3-input sorters rather than 2-input sorters, in order to decrease the number of stages of the filter [20]. The architecture of the *Max–Min Filter* is shown in Fig. 13. The *Max–Min Address Finder* unit finds the





X-(2N)-2	X-(2N)-1	X-(2N)
X-N-2	X-N-1	X-N
X	X-1	X-2

Fig. 12 Window pixels addressing, **a** procedure of window formation, **b** equivalent window located on image

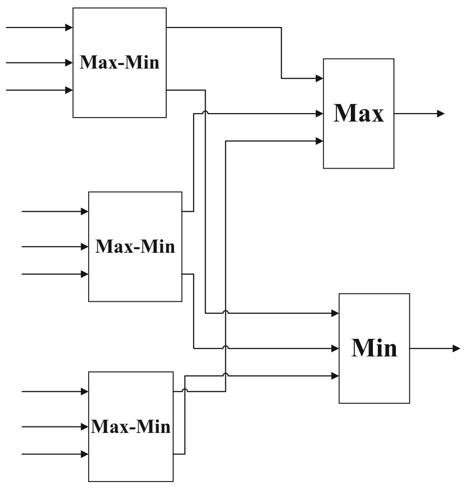


Fig. 13 Architecture of *Max–Min Filter*

addresses of the maximum and minimum pixels in the current window. The architecture of the *Noise Address Detector* is shown in Fig. 14.

3.1.3 Parallel/Serial blocks and Controller

The *Parallel/Serial* blocks and the *Controller* are the other building blocks of the *Noise Detection System*. Interconnection between them is shown in Fig. 15. The *Parallel/Serial* blocks receive the maximum and minimum pixels addresses of the window from the *Noise Address Detector* unit and serially feed them to the *RAM2* address port. The *Controller* block generates the required control signals for the proper operation of the *Parallel/Serial* blocks. As the detection process is finished, *Controller* triggers the *Noise Filtering System* to start its operation. As mentioned earlier, instead of using two memories, one for salt noise and the other for pepper noise mapping, only one RAM memory is used for mapping of both. The cells that are addressed by maximum addresses (i.e., *Parallel/Serial*1 block) are incremented, and those addressed by minimum addresses (i.e., *Parallel/Serial*2 block) are decremented. The stored values are represented in 2’s complement. Thus, those cells that hold “9” or “01001” are identified as salt

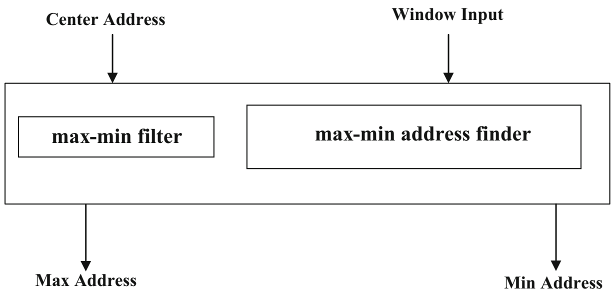


Fig. 14 Architecture of *Noise Address Detector*

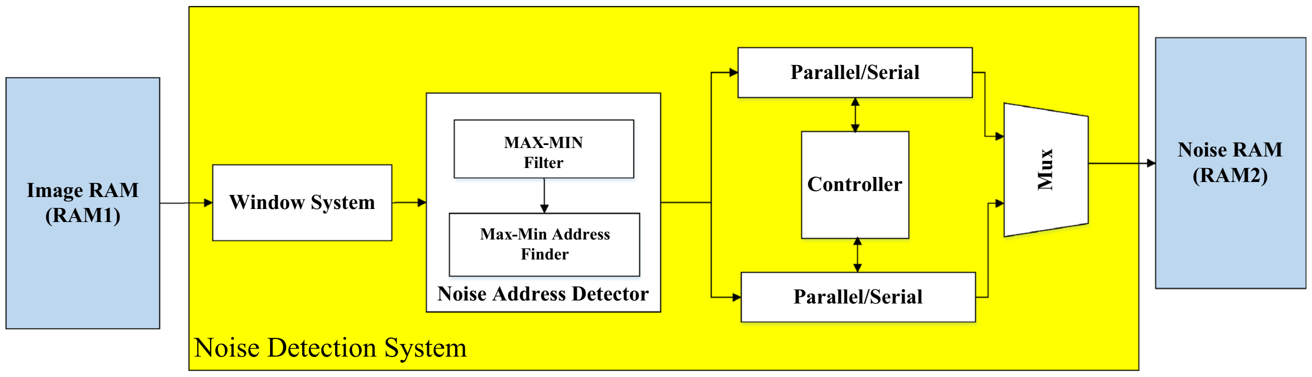


Fig. 15 Architecture of *Noise Detection System*

noise, and those hold "−9" or "10111" are detected as pepper noise; the other values are considered to be healthy pixels. Note that 5-bit words are required for the mapping sheet, and therefore, *RAM2* has 262144 × 5-bit capacity.

3.2 Noise Filtering System

Noise Filtering System, as also shown in Fig. 16, consists of *Controller*, *Window Size Selector*, *Noise Address Finder*, *Median Filter* and *Neighborhood Address Generator* units. The system is recursive, meaning that if some noisy pixels are located sequential, the next pixel utilizes the modified previous pixels. To find out the effect of using the modified previous pixels in comparison with using the corrupted ones, a MATLAB simulation is performed. Results show that for 20 and 50% noise densities, using modified pixels instead of corrupted ones, PSNR improvements of 4.7 and 3 dB are achieved, respectively. The details for *Noise Filtering System* sub-blocks are discussed below.

3.2.1 Window Size Selector

Window Size Selector block estimates the noise density percentage through calculating the number of noisy pixels, detected by *Noise Detection* System, divided by the number of image pixels. This unit, then, based on a pre-defined threshold that mentioned earlier, generates a signal that determines the size of window and configuration of the other building blocks of the *Noise Filtering System*, such as *Parallel/Serial* and *Median Filter*.

3.2.2 Controller

As shown in Fig. 16, *Controller* unit controls the sub-blocks of the *Noise Filtering System*. This unit is triggered at the end of detection phase and starts the filtering process. The *Controller* block reads the complete noise mapping memory (*RAM2*) cell by cell. If content of a cell equals "01001" or "10111," which means the related pixel is noisy, the *Controller* unit triggers its following module, i.e., *Neighborhood Address Generator*.

3.2.3 Neighborhood Address Generator

This unit receives the center pixel addresses from the *Controller* and generates the neighborhood addresses considering the image width. These addresses are used for setup the windows later.

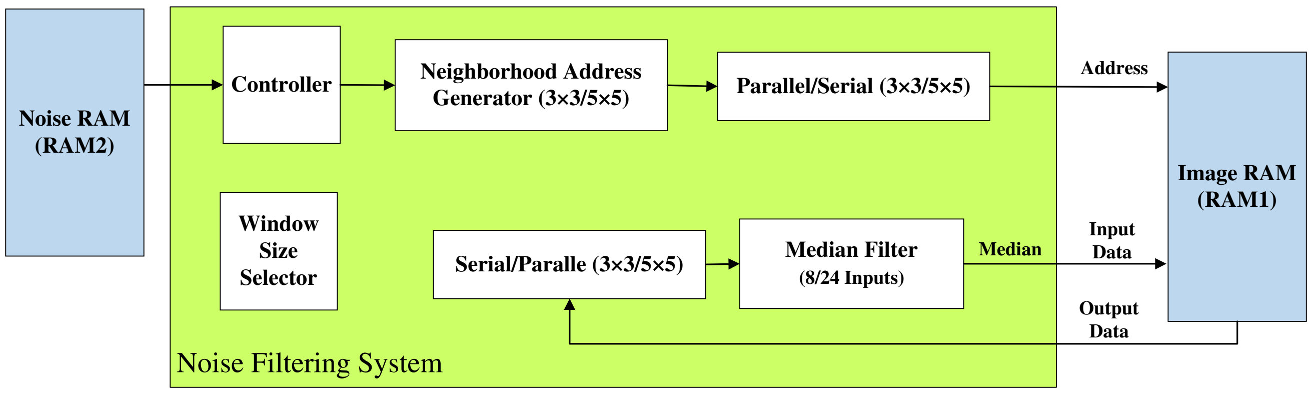


Fig. 16 Architecture of *Noise Filtering System*

3.2.4 Parallel/Serial and Serial/Parallel

The *Parallel/Serial* accepts the neighborhood addresses from the previous unit and feeds them into the *RAM1* address port. Then, the neighborhood pixels' values are read from *RAM1* and sent to the *Serial/Parallel* in order to simultaneously enter the *Median Filter* block.

3.2.5 Median Filter

After the window pixel values are received, *Median Filter* is only applied to the neighborhood pixels with exclusion of the center pixel. This filtering scheme results in a more precise value, since the corrupted center pixel is omitted from the window statistical population.

The core of the *Median Filter* is a constant two-stage sorting network, which means that the number of stages does not vary with number of inputs. The filter output is the median of the input sequence. For an even number of input items, output of this filter equals to average of the two median elements [21].

The *Median Filter* output is written back to *RAM1* at the center pixel address, where the noisy pixel is located. This procedure will continue until the last pixel of the image.

4 Implementation on FPGA

In this section, implementation of the system on the target FPGA is investigated. The implementation platform is EP4CE115F29C7 FPGA, which is a member of Altera Cyclone IV family.

In order to increase the operational frequency and reduce the disturbing glitches, the entire blocks of the systems became synchronized. This is performed by adding delay modules, to equalize the delays in the parallel paths, in all sub-blocks. In addition, high parallelism and interconnection are also exploited in the structure of the median filter so that it acquires the median of any number of inputs within fixed two clocks. This technique reduces the delay and increases the speed of the entire system.

In the following subsections, resource usage, power consumption and operational frequency of the implemented system are presented and compared with some previous work. Evidently, comparisons may not be very reasonable, because most of the algorithms implemented on FPGA are intended for low noise density suppression, and only few have claimed the implementation of medium or high noise suppression systems. In addition, FPGAs have a vast range of performances, power consumptions and costs, which cause the comparison to be less appropriate. Therefore, FPGA devices used for implementation as well as the resulted PSNR value are mentioned, wherever the information is available.

4.1 Resource usage

Resource usage of the proposed medium noise suppression system is shown and compared with previous work in Table 1. A fair comparison is not straightforward, since the implementations are different in terms of noise removal technique, image size, PSNR performance, operating frequency and most importantly FPGA platform. Also, the methods of reporting the hardware usage are not exactly the same.

The implementation of the proposed technique on the target platform requires 9954 4-input LUTs. For comparison, it is tried to find methods that can be applied to medium noise densities. However, two samples of low noise densities are added to the table, as well, which are located at the last two rows of the table. Please note that from the medium noise density techniques, Ref. [23] has introduced a low complexity method, which is useful for fast and low cost applications. It is implemented in 1397 logic cells only and has achieved a low PSNR of 26.86 dB. Ref. [24] has similar PSNR performance to us, but the hardware usage is not reported. Ref. [22], especially the pipelined version, could be used for an almost fair comparison. We have used more resources that is mostly due to the parallelizing techniques and as a result have achieved higher speed. Also, the size of image in our implementation is 512×512, which has resulted in larger buffer sizes and therefore larger number of FFs.

In addition to the hardware usage reported in the table, we have used 3,407,872 bits, from which $512 \times 512 \times 8 = 2{,}097{,}152$ bits are used to store the original and the denoised image (*RAM1*), and $512 \times 512 \times 5 = 1{,}310{,}720$ bits are used for storing the noisy pixels' map (*RAM2*).

4.2 Operating frequency

Timing analysis is a process for delay evaluation in logic circuits and digital systems, in order to find the conditions, in which the system can operate correctly and reliably. One of these conditions is maximum operating frequency. This frequency is determined and restricted by the critical path. Using *TimeQuest* slow 85 °C model, operating frequency of the system implemented physically on the target platform is measured, which is achieved at 93.27 MHz.

4.3 Power analysis

PowerPlay Power Analyzer of Altera is used for analysis of power consumption. Clock gating technique is utilized in order to reduce dynamic power of the system. This technique is used in all blocks except RAMs and the Controller blocks. Using clock gating in RAMs is avoided, due to some unsolvable timing issues. In each block, there is a

Table 1 Implementation results of our system and comparison with previous work

Ref. #	Method	Device (device family)	Hardware usage	Image size	PSNR (dB) (image, noise %)	Operating frequency (MHz)
Proposed system	Adaptive switching window	EP4CE115F29C7 (Altera Cyclone IV)	9954 4-inp. LUTs, 10293 Slice FFs, 5 IOs	512 × 512	33.14 (Lena, 50%) 35.15 (Boat, 50%)	93.27
[22] w/o pipeline	Edge preserving	XC3s500e-5-pq208 (Xilinx Spartan III-E)	2762 Slices, 2274 Slice FFs, 3743 4-inp. LUTs, 18 IOs	128 × 128	34.41 (Lena, 50%) 34.55 (Boat, 40%)	21.66
[22] with pipeline			3705 Slices, 2551 Slice FFs, 7242 4-inp. LUTs, 18 IOs			40.97
[23]	Low complexity (5 × 5 window)	EP2C20F484C7 N (Altera Cyclone II)	1397 Logic Cells, 8432 Slice FFs	512 × 512	26.86 (Lena, 50%)	93.76
[24]	Decision-based adaptive median (DBAM) filter	XC5VLX50T-2ff1136 (Xilinx Virtex V)	–	512 × 512	34.94 (Lena, 50%)	–
[26]	Median deviation based	Xc3 s 500e-4ft256 (Xilinx Spartan III-E)	2900 Slices, 849 Slice FFs, 4435 4-inp. LUTs	256 × 256	26.59 (Lena, 50%)	–
[15]	Adaptive median filter (AMF)	EPF10 K200SRC240-1 (Altera APEX 20K PLD)	9972 LEs, 52 IOs	260 × 244	37.43 (Cafe, 5%)	65
[25]	Adaptive median filter (AMF) (5 × 5 window)	XC5VLX50T-2ff1136 (Xilinx Virtex V)	1352 Slices, 2504 6-inp. LUTs, 192 FFs, 16 IO Bocks	–	38.41 (Lena, 5%)	–

combinational logic that declares the status of idleness. This status is used for clock gating management. In this way, modules, which are idle, become clock gated. The Altera Standard Clock Gating Scheme is exploited to avoid glitches. The power consumption distribution of the system is shown in Table 2.

4.4 Physical implementation

The proposed system is physically implemented on Altera's DE2-115 board. Figure 17a, b depicts the input 20 and 60% density noisy images, and Fig. 17c, d shows their respective output denoised images. Physical implementation with a VGA interface display is shown in Fig. 18. For the Boat image with 512 × 512 pixels and 50% noise density, after applying the algorithm, a PSNR of 35.15 dB is achieved, and the process is finished in 12 ms.

A summary of the PSNR performance of the system is reported in Table 3.

5 Discussion

As mentioned earlier, the denoising operation improves the picture quality, and therefore, it is necessary before doing other higher-level image processing. In order to illustrate the effect of impulse noise on other processes, such as edge detection, 20% impulse noise is added to the original image as shown in Fig. 19. The Sobel filter is applied to both corrupted image and the denoised one. As shown, even for low-density additive impulse noise, denoising is vital, before edge detection.

The system that is implemented on FPGA is assumed for denoising a 512 × 512 pixel image. For larger size images, larger amount of memory is required and also more time is needed for denoising. It is expected that the memory size and processing time increase linearly with size of the input image. For example, for a 1024 × 768 pixel image, the processing time is expected to be tripled or 36 ms.

An improvement, which can be applied to the FPGA implementation, is that if low memory usage is a requirement, the size of noise RAM (RAM2) can be reduced, by combining the two operating phases, detection and filtering. When the *Noise Detection System* is working on the 3 × 3 window around pixel (x, y), all pixels of rows 1 to $x - 2$ have undergone noise detection, and the noisy ones

Table 2 Power analysis for a medium noise density system

	Power (mW)
Core dynamic power	75.15
Core static power	103.12
I/O blocks power	31.46
Total power	209.73

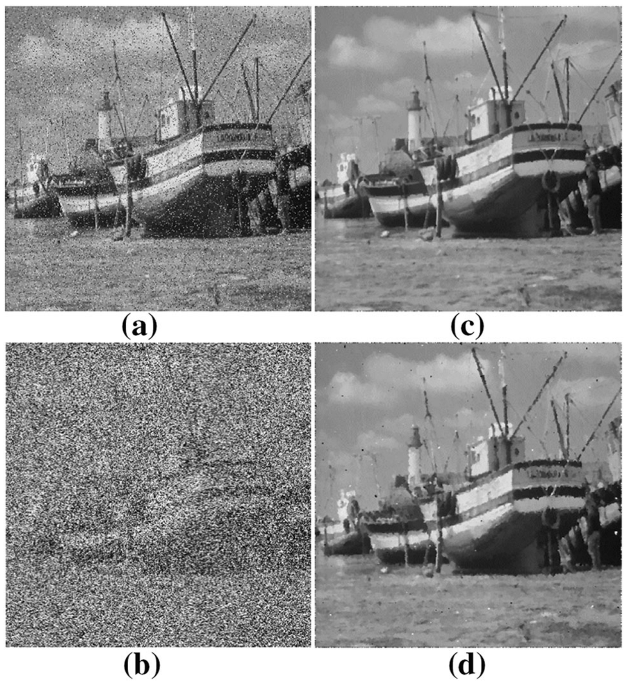


Fig. 17 The results of applying the algorithm, implemented on an FPGA, on noisy images. **a**, **b** The input noisy images with 20 and 60% noise densities. **c**, **d** The related output denoised images

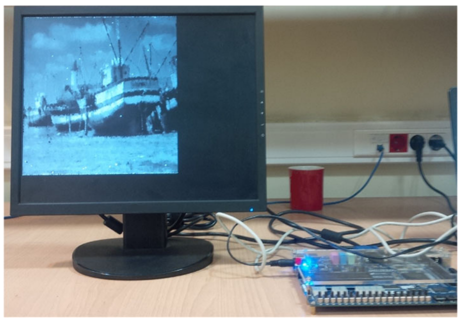


Fig. 18 Physical implementation of the system on FPGA

are flagged for filtering. Therefore, the *Noise Filtering System* can operate on those pixels and suppress their noises. It means that filtering can start after detection reaches the third row of the image. With this technique, also the propagation delay (the time for a pixel going through the noise suppression system from input to the output) is significantly reduced. However, those enhancements are achieved at the cost of higher system complexity.

6 Conclusions

In this paper, first, a highly efficient algorithm called ASM, which is used for image impulse noise suppression, was reviewed and some modifications were proposed on it for low noise densities. Then, an architecture was introduced for the hardware implementation of the modified ASM algorithm. The system consists of two major sections of noise detection and noise filtering. The first section detects the noisy pixels in image and the latter suppresses only the corrupted ones. In this way, PSNR improved significantly [27–29]. This architecture has exploited pipelining and parallelism techniques in order to improve the performance for real-time applications. In the implementation level, memory resources have been significantly reduced, as well. The system has been physically implemented on an FPGA from Altera Cyclone IV family, using the DE2-115 evaluation board. The achieved operational frequency is 93.27 MHz. Experimental results confirm the functionality

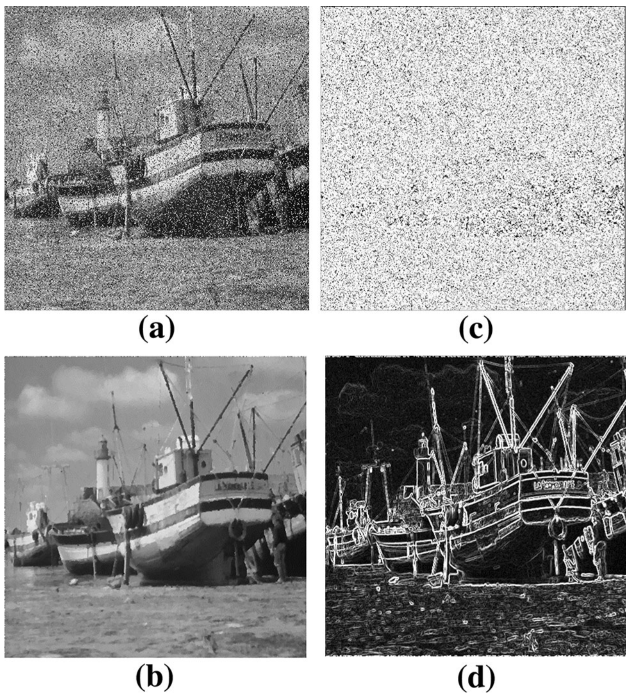


Fig. 19 Effect of noisy pixels on edge detection process, **a** image with 20% noise density, **b** the denoised image, **c**, **d** effect of applying edge detection on both images

Table 3 PSNR performance of the proposed system

Image	Noise density (%)	PSNR (dB)
Boat	30	38.86
	50	35.15
	70	30.19
Lena	30	37.76
	50	33.14
	70	27.53
Pepper	30	36.58
	50	32.15
	70	28.74

of the system. For visual verification, the processed image has been displayed through a VGA interface.

References

1. CCD Image Sensor Noise Sources: Application Note, 1st edn. Eastman Kodak Company, Rochester (2001)
2. Cumani, C.: Introduction to CCDs. In: Neutron Radiography, Fifth International Topical Meeting on Garching (2004)
3. Varghese, Gijesh, Wang, Zhou: Video denoising based on a spatiotemporal Gaussian scale mixture model. IEEE Trans. Circuits Syst. Video Technol. **20**, 1032–1040 (2010)
4. Jin, L., et al.: Quaternion-based impulse noise removal from color video sequences. IEEE Trans. Circuits Syst. Video Technol. **23**, 741–755 (2013)
5. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 2nd edn. Prentice-Hall Inc, New Jersey (2002)
6. Guo, Q., et al.: An efficient SVD-based method for image denoising. IEEE Trans. Circuits Syst. Video Technol. **26**, 868–880 (2016)
7. Yang, J., et al.: Image and video denoising using adaptive dual-tree discrete wavelet packets. IEEE Trans. Circuits Syst. Video Technol. **19**, 642–655 (2009)
8. Malinski, Lukasz, Smolka, Bogdan: Fast averaging peer group filter for the impulsive noise removal in color images. J. Real-Time Image Proc. **11**, 427–444 (2016)
9. Malinski, L., Bogdan S.: Fast adaptive switching technique of impulsive noise removal in color images. J. Real-Time Image Process. 1–22 (2016). doi:10.1007/s11554-016-0599-6
10. Chatterjee, P., Milanfar, P.: Is denoising dead? Image Process. IEEE Trans. **19**(4), 895–911 (2010)
11. Ko, S.-J., Lee, Y.H.: Center weighted median filters and their applications to image enhancement. Circuits Syst. IEEE Trans. **38**, 984–993 (1991)
12. Wang, Z., Zhang, D.: Progressive switching median filter for the removal of impulse noise from highly corrupted images. Circuits Syst II Analog Digit. Signal Process. IEEE Trans. **46**, 78–80 (1999)
13. Windyga, P.S.: Fast impulsive noise removal. Image Process. IEEE Trans. **10**, 173–179 (2001)
14. Chen, T., Ma, K.-K., Chen, L.-H.: Tri-state median filter for image denoising. Image Process. IEEE Trans. **8**, 1834–1838 (1999)
15. Fabijanska, A., Sankowski, D.: Noise adaptive switching median-based filter for impulse noise removal from extremely corrupted images. Image Process. IET **5**(5), 472–480 (2011)
16. Eng, H.L., Ma, K.K.: Noise adaptive soft-switching median filter. Image Process. IEEE Trans. **10**(2), 242–251 (2001)
17. Andreadis, I., Louverdis, G.: Real-time adaptive image impulse noise suppression. Instrum. Meas. IEEE Trans. **53**(3), 798–806 (2004)
18. Hwang, H., Haddad, R.: Adaptive median filters: new algorithms and results. Image Process. IEEE Trans. **4**(4), 499–502 (1995)
19. Hu, Y., Ji, H.: Research on image median filtering algorithm and its FPGA implementation. In: Intelligent Systems. GCIS'09. WRI Global Congress on IEEE (2009)
20. Nieminen, A., Neuvo, Y.: Comments on theoretical analysis of the max/median filter" by GR Arce and MP McLaughlin. Acoust Speech Signal Process. IEEE Trans. **36**, 826–827 (1988)
21. Baddar, S.W.A.H., Batcher, K.E.: Designing Sorting Networks: A New Paradigm. Springer, New York (2012)
22. Jayanthi Sree, S., Ashwin, S., Aravind Kumar, S.: Edge preserving algorithm for impulse noise removal using FPGA. In: Machine Vision and Image Processing (MVIP), 2012 International Conference on IEEE (2012)
23. Matsubara, T., Moshnyaga, V.G., Hashimoto, K.: A FPGA implementation of low-complexity noise removal. In: Electronics, Circuits, and Systems (ICECS), 17th IEEE International Conference on IEEE (2010)
24. Kamarujjaman, M.M., Maitra, M.: An efficient FPGA based de-noising architecture for removal of high density impulse noise in images. In: IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) (2015)
25. Mukherjee, M., Maitra, M.: Reconfigurable architecture of adaptive median filter—an FPGA based approach for impulse noise suppression. In: Computer, Communication, Control and Information Technology (C3IT), 2015 IEEE Third International Conference, pp. 1–6
26. Anand, M., Narasimha, Y.: Removal of salt and pepper noise from highly corrupted images using mean deviation statistical parameter. Int. J. Comput. Sci. Eng. **5**(2), 113–119 (2013)
27. Fahmy, S.A., Cheung, P.Y., Luk, W.: Novel FPGA-based implementation of median and weighted median filters for image processing. In: Field Programmable Logic and Applications, International Conference, pp. 142–147 (2005)
28. Shi, P., Ward, R.K.: A neural network implementation of the median filter. In: Communications, Computers and Signal Processing, IEEE Pacific Rim Conference, pp. 513–516 (1989)
29. Gil, J., Werman, M.: Computing 2-D min, median, and max filters. In: Pattern Analysis and Machine Intelligence, IEEE Transactions, pp. 504–507 (1993)

Parham Taghinia Jelodari received B.E. degree in Electronics Engineering from Babol Noshirvani University of Technology, Mazandaran, Iran, and an M.S. degree in Electronics Engineering from University of Tehran, Tehran, Iran. He is currently researching in Hardware Implementation of Image and Video Processing Lab of University of Tehran, Tehran, Iran. His research interests include image and video processing, neural networks, embedded systems and implementation of DSP systems.

Mojtaba Parsa Kordasiabi received B.E. degree in Electronics Engineering from Sharif University of Technology, Tehran, Iran, and an M.S. degree in Electronics Engineering from University of Tehran, Tehran, Iran. He is now senior researcher in Hardware Implementation of Image and Video Processing Lab of University of Tehran. His research interests include VLSI, neural networks, custom implementation of DSP systems particularly image and video processing systems.

Samad Sheikhaei received the B.Sc. and M.Sc. degrees in Electrical Engineering from Sharif University of Technology, Tehran, Iran, in 1996 and 1999, respectively, and the Ph.D. degree from the University of British Columbia, Vancouver, BC, Canada, in 2008. He was engaged in Research and Design Engineering at Sharif University of Technology. He also worked in industry for a couple of years. In September 2009, he joined the Department of Electrical and Computer Engineering at University of Tehran, Iran, where he is currently an Assistant Professor. His research interests are analog, mixed signal and RF integrated circuits design, as well as VLSI/FPGA implementation of signal processing algorithms.

Behjat Forouzandeh received M.S. degree in the field of Electrical Engineering from University of Tehran, Tehran, Iran, in 1980 and her Ph.D. in the field of Electrical Engineering from the University of Birmingham, UK, in 1997. She joined the Department of Electrical and Computer Engineering at University of Tehran, Iran, where she is currently an Associate Professor and involved in various courses and laboratories such as analog/digital electronics, SOI and quantum computing and VLSI. Her current research interests include analog and mixed signal circuits, SOI devices, quantum computing design and implementation. She is also author or coauthor for plenty of journal and conference papers in various fields of Electronics and Hardware Engineering.