A Low-Complexity and Low Power Design of 2D-Median Filter

Takeaki Matsubara[1], **Vasily G. Moshnyaga**[2], and **Koji Hashimoto**[3], Non-members

ABSTRACT

Impulse noise removal is a very important preprocessing operation in many computer vision applications. Usually it is accomplished by median filter with excessive sorting and therefore large power. This paper presents a new design of 2D median filter that utilizes a simple conditional filtering technique, executes fewer computations than related designs while achieving superior image quality. Experimental FPGA implementation of the proposed filtering scheme is compact, fast and low-power consuming.

Keywords: Impulse Noise, Median Filter, FPGA Implementation, Image Processing

1. INTRODUCTION

Median filter [1] is a non-linear digital filter widely used in image processing for image smoothing and suppression of impulse noise, which frequently corrupts images during picture acquisition or transmission. In general two types of impulse noise exist: fixed-valued impulse noise and random valued impulse noise. The fixed value impulse noise [2] is usually reflected by a pixel which has either a minimum or a maximum value in gray-scale image. In contrast, the values of randomvalued noisy pixels are distributed uniformly in the grayscale image within the range of [0, 255]. The median filter removes impulse noise signals by changing the luminance value of the target pixel with the median value of those pixels in the filtering window. However, as the number of corrupted pixels in the image increase, the median filter produces poor results. Namely, it blurs image details and causes loss of the useful information in the image. Besides, since most median filters require sorting or inserting/deleting procedures, they become very computationally expensive.

Over the years, various adaptive techniques for impulse noise reduction have been proposed [3-20]. The common idea is to split the noise processing into two parts: noise detection and noise removal by filtering. The incoming data is checked and if a noisy pixel is found, the adaptive filter is applied to repair the corrupted pixel. Otherwise, the original pixel is kept. In comparison to median filtering, the adaptive techniques do not process the noise-free pixels and so reduce the computational load.

In general, the adaptive techniques proposed for impulse noise removal can be classified as lower complexity techniques and higher-complexity techniques. The lower complexity techniques [3-18] use fixed size windows and simplified computations in order to achieve high-speed processing. The higher complexity techniques [19-22] target excellent visual quality by adaptively enlarging window sizes or increasing computing iterations. In this paper we focus only on the low complexity techniques because of their implementation simplicity, low processing time and low power consumption.

Existing lower-complexity noise removal techniques differ by detection of noise pixels and their de-noising. In [3],[4], weights are applied to control filtering while preserving features of given shapes and sizes. [5],[6],[7] achieve fast filtering by single thresholding and so limit themselves to lower noise density levels. The combination of simple thresholding with center-weighted median filtering is given in [8][9]. Zhang and Karim[10] use 1-D Laplacian operators to compute four 5x5 convolution kernels and apply them to separate impulses from edges. Jiang [11] propose to truncate each noise pixel by the maximal or minimal values of its surrounding pixels. Andreadis and Louverdis [12] multiply the minimal and the maximal values of pixels in the search window by a predefined real number and use them as the noise thresholds. Aizenberg and Butakoff [13] advocate identifying the noise pixel based on both rank and absolute value. Their differential rank impulse detector defines a pixel (i,j) noisy if $(R(i,j)\leq k)$ $(R(i,j)\geq N-k+1)\wedge(F(i,j)\geq Q)$, where R(i,j) is the rank of pixel (i,j), i.e. its position from 1 to N; k and Q are two thresholds; and $\vee$ and $\wedge$ are disjunction and conjunction operators, respectively. Alpha-trimmed mean based impulse noise detection is reported by Luo in [14], [15]. A pixel is considered noisy if it matches one of the peak locations in the image histogram. In this case, the detector calculates the minimum of absolute differences between the pixel value and the values of its eight neighbors and generates a fuzzy map S(i,j), which indicates how much each pixel looks like

Manuscript received on March 2, 2011 ; revised on May 12, 2011.

[1,2,3] The authors are with the Department of Electronics Engineering and Computer Science, Fukuoka University, 8-19-1 Nanakuma, Jonan-ku, Fukuoka 814-0180, Japan Tel/Fax: (+81) 92-801-0833 Email: matsubara, vasily, khashi@fukuoka-u.ac.jp

The work was sponsored by The Ministry of Education, Culture, Sports, Science and Technology of Japan under the Knowledge Cluster Initiative (The Second Stage) and Grant-in-Aid for Scientific Research (C) No.21500063

an impulse noise. The value of pixel (i,j) is then replaced by a linear combination of its original value F(i,j) and the median value of its neighbors. Srinivasan and Ebenezer[16] sort pixels in the window to obtain minimum, maximum and

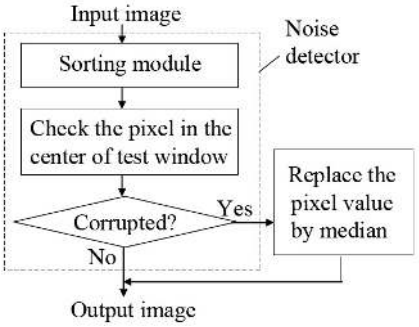

Fig.1: *An overview of the proposed technique*

mean values and then use them to make decisions on weather to remove the corrupted pixel either by median or by its neighboring pixels. Ibrahim, et al [17] modify the histogram-based method [15] by counting the number of noisy pixels at the noise detection stage in order to estimate the level of impulse noise that corrupts the image. The obtained level is then used for the window size control at the noise cancelation stage. The edge preserving image de-noising proposed by Chen et al, [18] detects noise pixels based on the minimal and maximal values computed for the current and the previously processed windows, respectively. To preserve edges, the detector calculates 12 absolute differences and adds them in pairs to compute directional differences around the central pixel F(i,j). Then it determines the minimum value along those directional differences, checks four denoising conditions, and estimates eight arithmetic functions (16 additions, 8 shifts (divisions) and 8 comparisons in order to reconstruct the corrupted pixel.

Despite difference in implementation, all existing solutions have one feature in common. Namely their efficiency strongly depends on the thresholds used for noise detection. On the one hand, fixed or image independent thresholds yield fast results but lead to loss of detail and smoothing of edges. On the other hand, the use of dynamic or image-dependent thresholds reduces misdetection but requires computationally expensive operators (e.g. multiplications, divisions, etc.) and large memory resources which increase cost, processing time and power consumption of implementation hardware. Therefore dynamic thresholds, which can be computed by simple and compact hardware, are very important.

In this paper, we present a novel lower-complexity adaptive filter that achieves high quality processing under low cost requirements on image de-noising hardware. The paper's contribution is twofold. The first one is noise detector which exploits dynamic thresholds, optimized to reduce the number of operations while maintaining the high visual quality. The second contribution is hardware implementation of adaptive filter, which in comparison to related designs requires less resources and power.

The rest of the paper is organized as follows. The next section presents the proposed technique. Section 3 describes hardware architecture. Section 4 shows experimental results. Section 5 gives conclusion and outlines work for the future.

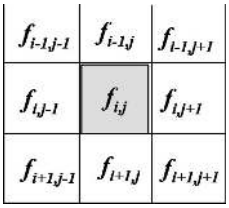

Fig.2: *The relative position of current processing pixel (i,j)*

2. THE PROPOSED NOISE DETECTOR

2.1 Main idea

The proposed technique aims at removing random-valued impulse noise. Similarly to the other adaptive filtering methods, we assume that the filter contains two components: noise detector and noise remover, as shown in Fig.1. The noise detector determines whether the pixels are corrupted by the impulse noise or not. Each noisy pixel is processed by median filtering for reconstruction. Otherwise, there is no fundamental reason to modify the value of a non-noisy pixel, so the filtering is skipped. Below we discuss the technique in details.

2.2 The noise detector

The noise detector uses two dynamically defined conditional thresholds (T_{LOW}, T_{HIGH}) to distinguish corrupted pixels from the noise-free pixels. Let f_{ij} be the value of the pixel with coordinates (i,j) and W(i,j) the set of pixels that surround the (i,j) within the test window of (N×N) pixels in size. Fig.2 shows the relative position of the pixel (i,j) and its (3×3) test window.

In order to consider edges we propose to calculate thresholds based on the difference between the pixel of interest and its closest neighbors in the variation series (the neighbor is chosen from the interval between the pixel of interest and the median). Namely, our noise detector sorts the values of pixels within W(i,j) in ascending order and evaluates values of three image pixels $(i, j-1)(i, j)(i, j+1)$ in the middle. Let the values of these pixels be $f4_{i,j}, f5_{i,j}, f6_{i,j+1}$, and $f4_{i,j} \leq f5_{i,j} \leq f6_{i,j}$ That is $f5_{i,j}$ is the median value of N^2 values in W(i,j). Then the thresholds T_{LOW} and T_{HIGH} are defined as following:

Table 1: *The ranges of α and β at which the PSNR difference from PSNR MAX was less than 0.5dB*

Image	Noise 10%			Noise 20%		
	MAX PSNR	α range	β range	MAX PSNR	α range	β range
Bridge	38.69 (α=34,β=50)	27~39	40~150	36.01 (α=34,β=40)	18~34	34~70
Temple	36.60 (α=31,β=59)	28~41	43~138	33.22 (α=30,β=63)	21~38	38~80
Airplane	36.69 (α=31,β=134)	29~32	98~150	32.52 (α=30,β=135)	25~32	68~150
Boat	35.56 (α=33,β=63)	27~36	48~147	32.41 (α=29,β=70)	23~35	50~91
Parthenon	38.26 (α=36,β=68)	29~36	59~97	34.68 (α=31,β=58)	25~35	44~87
Barbara	28.49 (α=42,β=93)	39~59	69~150	26.23 (α=38,β=87)	30~46	52~144
Lena	37.35 (α=28,β=75)	20~34	54~118	34.52 (α=39,β=58)	29~43	43~84
Baboon	27.44 (α =55, β =94)	35~60	82~150	24.67 (α =40, β =86)	35~60	60~150
Pentagon	31.90 (α =37, β =121)	30~41	70~150	28.82 (α =30, β =90)	30~60	50~150

Table 2: *The PSNR values observed for different α and β on tested images (512×512 pixels in size)*

	Noise 10%							Noise 20%						
	MAX PSNR	α =30 β =70	α =30 β =80	α =35 β =70	α =35 β =80	α =40 β =70	α =40 β =80	MAX PSNR	α =30 β =70	α =30 β =80	α =35 β =70	α =35 β =80	α =40 β =70	α =40 β =80
Bridge	38.69	38.37	38.35	38.37	38.35	38.08	38.01	36.01	35.52	35.27	35.3	35.06	34.8	34.56
Temple	36.6	36.54	36.51	36.54	36.5	36.43	36.37	33.22	33.19	33.14	32.94	32.89	32.75	32.69
Airplane	36.69	35.74	35.84	35.21	35.28	33.68	33.74	32.52	32.2	32.17	31.56	31.54	30.21	30.19
Boat	35.56	35.46	35.48	35.44	35.42	35.11	35.1	32.41	32.4	32.3	32.21	32.12	31.74	31.67
Parthenon	38.26	37.97	37.95	38.23	38.14	36.46	36.39	34.68	34.56	34.46	34.43	34.3	32.71	32.59
Barbara	28.49	27.25	27.25	27.74	27.76	28.08	28.13	26.23	26.0	26.0	26.15	26.16	26.15	26.19
Lena	37.35	37.01	37.01	37.3	37.33	36.92	36.92	36.1	35.69	35.63	35.91	35.83	35.87	35.8
Baboon	27.44	25.25	25.3	25.8	25.83	26.31	26.41	24.67	23.95	23.99	24.29	24.35	24.59	24.66
Pentagon	31.9	31.41	31.46	31.71	31.77	31.65	31.74	28.82	28.74	28.78	28.76	28.79	28.5	28.52
Average	-	33.81	33.83	33.89	33.89	33.48	33.49	-	31.36	31.30	31.28	31.23	30.81	30.76
Δ	-	0.63	0.61	0.55	0.54	0.95	0.94	-	0.27	0.32	0.34	0.4	0.81	0.86

$$T_{HIGH} = \begin{cases} f6_{i,j} + \alpha, if f6_{i,j} + \alpha < f5_{i,j} + \beta \\ f5_{i,j} + \beta, \text{otherwise} \end{cases} \quad (1)$$

$$T_{LOW} = \begin{cases} f4_{i,j} - \alpha, if f4_{i,j} - \alpha < f5_{i,j} - \beta \\ f5_{i,j} - \beta, \text{otherwise} \end{cases} \quad (2)$$

where α and β are empirically predefined parameters, which set the difference in brightness between the pixel (i,j) and its close neighbors in the variation series. The upper conditions in (1) and (2) determine the thresholds in the presence of an edge; the low ones define the thresholds in its absence.

If the detector finds that the value $f_{i,j}$ of pixel (i,j) is larger than T_{HIGH} or lower than T_{LOW}, it sets a binary flag S to one. Otherwise, the S is 0. That is,

Thus, the flag ($S = 1$) points out that the pixel (i,j) is corrupted. If $S = 0$, the pixel has a noise-free value.

2.3 The Median Filter

The median filter is activated if and only if the flag S is set. The filter replaces the corrupted value of current processing pixel (i,j) with the median value of those pixels in W(i,j). In contrast to traditional median filters [1] as well as the adaptive center-weight filters[3-4], which replace the central pixel intensity value $f_{i,j}$ in the

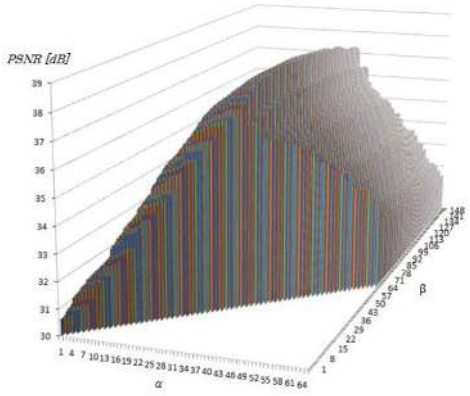

Fig.3: *PSNR variation with α and α*

window N times, our filter replaces $f_{i,j}$ only once, namely if S=1. Otherwise, the pixel value remains unchanged. This allows the noise filter to be tuned to

Fig.4: *An illustration of visual quality: (from left to right) original image; 20% noisy image; standard median filter; the proposed technique on standard test images (from top to bottom): Lena, Bridge, Boat, Parthenon, Airplane*

the expected noise characteristics (unlike the center weighted median filter) and also provides significantly less change to the original noiseless pixels (unlike the traditional median filter). It should be noticed that the proposed conditional median filter turns to the standard median filter for $T_{LOW} = T_{HIGH}$.

2.4 Evaluation

To examine the properties of the proposed noise detection technique, we performed several tests. The first test was dedicated to computing the parameters (α and β) of the thresholds (T_{HIGH} and T_{LOW}), which lead to the best picture quality in terms of Peak-Signal to Noise ratio (PSNR) and mean squared error (MSE). We used nine standard gray-scale test images (Lena, Barbara, Bridge, Baboon, Airplane, Parthenon, Pentagon, Temple and Boat) each of each was evaluated at 512×512 and 256×256 pixels in size as well as at 10% and 20% density of impulse noise, respectively. The noisy images were then repeatedly processed by the proposed technique (3×3 window size) with α varying from 0 to 65, while for each value of α the value of β was changing from 0 to 150. Fig.3 illustrates the PSNR variation with α and β observed for Lena image (512×512 pixels in size, 10% noise). As one can see, the PSNR peaks at $20 < \alpha < 34$ and $54 < \beta < 118$ with the maximum at α=28 and β=118. The results also revealed that the values of α and β which bring peak PSNR depend on both the image content and the noise density but do not depend on the image size. (The PSNR variation for 512×512 and 256×256 image sizes had the same variation pattern).

Table 1 summarizes the results obtained for images of 512×512 pixels in size in terms of the maximal PSNR and ranges of α and β at which the PSNR variation from the maximum value was less than 0.5dB. Based on these ranges, we derived combinations of α and β and selected those, which maximized PSNR across all tested images. Table 2 exemplifies the results in terms of PSNR, the average PSNR value for corresponding α and β and the difference (Δ) with MAX PSNR accumulated over all the test images. Thus we selected α=35 and β=80.

Table 3: *PSNR and MSE for different filter sizes*

Parameter	10%noise		20%noise	
	PSNR(dB)	MSE(db)	PSNR(dB)	MSE(db)
MF(3×3)	15.05	2032.3	12.07	4040.08
Our(3×3)	37.33	11.99	34.32	24.04
MF(5×5)	29.7	69.62	28.33	95.59
Our(5×5)	34.56	22	31.57	45.25

Having the values of α and β set, we compared the performance of the proposed filter with conventional median filter [2]. Table 3 shows the PSNR and MSE values after processing the image Lena (512×512 pixels in size) corrupted by 10% and 20% noise by standard median filter (MF) and the proposed technique for two window sizes: 3×3 and 5×5, respectively. Notice, the quality of results (in terms of PSNR and MSE) produced by the proposed technique is considerably better than those of the standard median filter.

Fig.4 illustrates the visual quality of corresponding images for 3×3 window size and 30% noise density. Because our method relies on dynamically computed image-dependent thresholds, it is able to preserve edges, which are blindly blurred by traditional median filter. Clearly, the picture quality achieved by the proposed method is close to the original one.

Next, to access the effectiveness of the proposed technique, we compared the results obtained by our technique with those produced by related methods. Totally five de-noising methods have been tested:
1) standard median filter (MF) of size (3x3) [1];
2) the differential rank impulse detector (DRID) [13];
3) the alpha-trimmed mean based method (ATMBM) [14],
4) the edge-preserving image de-noising method (EPID) [17] and
5) the proposed technique.
The threshold parameters of the related methods were set as described in the corresponding papers.

Table 4 shows the PSNR values of the images with noise densities varying from 10% to 50%. As one can see, our technique achieves better results than related methods, yielding only to a more computationally expensive edge preserving (EPID) method [17] by less than a half dB. However, if we consider the visual quality, this PSNR difference is almost invisible (see Fig.5).

Table 4: *Comparison on PSNR*

Noise ratio	MF	DRID	ATMBM	EPID	Our
10%	32.65	36.95	36.88	37.85	37.33
20%	31.28	32.86	34.28	34.58	34.32
30%	29.51	28.94	31.95	32.15	31.98
40%	27.59	25.28	29.87	30.03	29.94
50%	25.54	23.73	26.35	27.25	26.86

Table 5: *Comparison on computational complexity*

Parameter	MF	DRID	ATMBM	EPID	Our
Comp/abs	28	144	144	59	40
Add/sub	0	31/70	27/47	23	8
Mult	0	0	16	0	0
Proc.time	0.22	1.32	3.03	0.68	0.33
Buffer size	2lines	512×512	512×512	2lines	2lines

Table 5 compares the related methods in terms of required operations, processing time and the buffer size. Here, *comp* shows comparisons; *abs* - absolute value computations; *add* additions; *sub* subtractions, and *mult* multiplications. The processing time was obtained from the PC with 2.8GHz Pentium CPU

and 512MB memory. Based on the results, we conclude that our noise detection technique requires less arithmetic operations than the DRID, ATMBM, and EPID, i.e. has less computational complexity, while utilizing the minimum number of memory buffers.

Fig.5: *The Lena image produced by [17](top) and by our technique (bottom)*

3. HARDWARE IMPLEMENTATION

Fig.6 outlines hardware architecture for implementing the proposed technique. The architecture consists of three major blocks: register bank, sorting unit, and threshold generation and noise detection unit. In every processing step, the architecture reads image data from memory (not shown for simplicity) and shifts two line buffers to the right, placing the incoming pixel X and its two neighboring pixels into the register bank and the shifter, while two of de-noising pixels are written into the line shifters and memory, respectively. With each odd clock cycle the shifter sends a new pixel into the sorting circuit, which takes two clock cycles to sort all N×N pixels in the filtering window and detect the median and its left and right neighbors (see Section 2.2). Based on these three values, the threshold generation logic produces signal S to control the multiplexor and replace the pixel (i,j) currently located in the center of the register bank by new (median) value or keep the pixel value unchanged. Below we briefly describe each of the architectural blocks in more details.

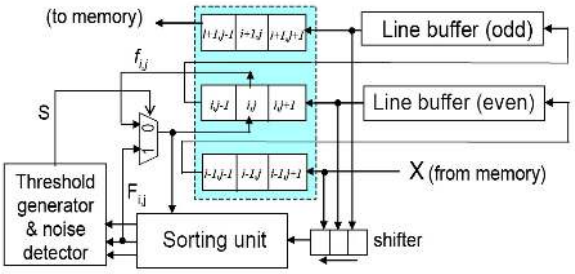

Fig.6: *The proposed hardware architecture*

3.1 Register bank

The register bank consists of N×N registers to store all pixel values of the current window. When the window is shifted from the current location to the next one, only N new values are read into the Register Bank (RB) and the rest (N-1)×N pixel values are shifted to their right registers, respectively. Eventually, N values are loaded into the shifter in parallel. The shifter operates at N times higher clock frequency than the register file, moving the received data one position to the right in every clock cycle. With each data move one pixel sample enters the sorting unit. Thus its takes N machine cycles (or N×N clock cycles of the shifter) to process all pixels in the window.

3.2 Sorting Unit

To implement sorting in hardware, we use 1D-structure proposed in [23]. The circuit consists of $M = N^2$ cascaded blocks, one for each window rank, as shown in Fig.7. Each block i is composed of one n-bit register (R_i), one k-bit ($k = log_2 M$) counter (P_i), n-bit comparator (C) and a simple data-transfer logic. All blocks are connected to the Reset line and the global input, X, through which they receive the incoming sample. The data-transfer logic allows blocks to receive and transmit the content of their R and P values from/to their neighbors. The registers $R_1, R_2, \ldots, R_M$ store samples in descending order; so at time t the maximum value is always at the left (in R_1); the minimum value at the right (in R_M), and the median is in the register R_i. We assume the sorting unit runs at the twice higher clock frequency than the shifter: i.e. it performs two clock cycles for each clock cycle of the shifter. At the odd clock cycle, the sorter increments all the counters (P_j) to maintain sample aging while removing from the window (W) that datum whose value exceeds M. At the even clock cycle, it compares the input datum, x_t, to all samples stored in the registers R_j, and moves those samples, whose values are less than x_t, to the right while putting the input datum, x_t, into the vacant place within the already ordered sequence.

To illustrate the circuit operation, assume that registers R_1, R_2, R_3, R_4 and R_5 store 152, 140, 135, 31, and 0, respectively, while counters P_1, P_2, P_3, P_4 and

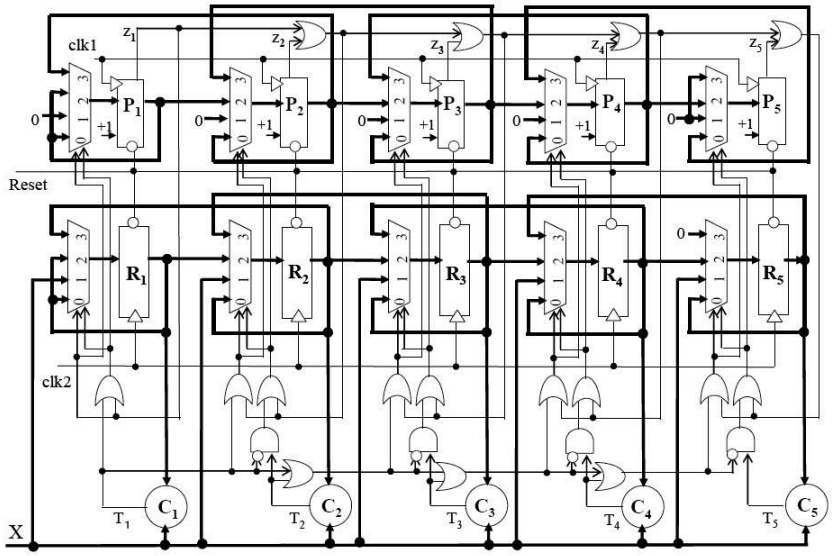

Fig.7: *Logic structure of the sorting unit (M=5)*

P_5 have 1, 4, 3, 2, and 0, respectively. Let the incoming input sample is ($X = 145$). At any odd clock cycle, the counters are incremented. The counter P_2 reaches 5, which is the limit for $M = 5$, and overflows sending the request signal z_3 to all the blocks on its right. By receiving this signal, the blocks 3,4 and 5 move their values to the adjacent blocks on the left, while the values of R_5, P_5 are reset to zero. Thus R_1 becomes 152, R_2=135, R_3=31, P_4=0, R_5=0, P_1=2, P_2=4, P_3=3, P_4=1, P_5=0 while the aged sample 140 is removed from the window.

At any even clock cycle, the input sample is compared to all values stored in the registers. The comparators (Ci) produce true signals T_i=1 if $X > R_i$; otherwise T_i=0. Thus, since $X = 145$ is larger than the value of the register R_2 but lower than that of R_1, the signal T_1 generated by C_1 becomes 0 while T_2 (from C_2) is 1. These signals enforce the multiplexor in front of R_2 to select input 1 while all the multiplexors at the right side of R_2 select input 2. Thus the contents of registers R_2-R_5 is moved one position the right and the sample X is written to the R_2. Due to this partial data movement left and right, the circuit preserves the sample ordering obtained at the present cycle, while resorting only the new incoming sample at a new clock cycle.

3.3 Threshold generator and noise detector

Fig.8 shows the internal structure of the threshold generation and the noise detection unit. Here C denotes comparators and + denotes adders. The circuit implements equations (1)-(3) in one clock cycle, producing the signal S=1, when the pixel (i,j) is noisy and S=0, otherwise.

Based on this signal, the multiplexer in Fig.6 selects either the median or the original pixel value and writes it to the location (i,j) in the register bank, as well as to the sorting circuit, to replace the noisy pixel with the median value. After that the content of the RB is shifted left and the processing repeats.

Overall the proposed hardware architecture shown in Fig.6 takes six cycles of internal (sorting) clock to process three new pixels and two extra clock cycles to replace the noisy pixel in the sorting window.

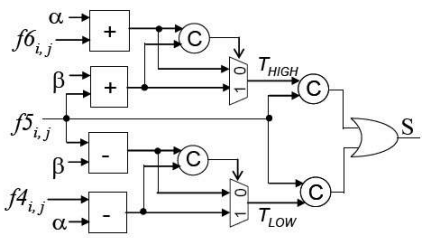

Fig.8: *The threshold generator and noise detector*

3.4 Hardware evaluation

To evaluate the filtering hardware, we experimentally designed two FPGA implementations by using Altera Design Tools: one is the conventional median filter, proposed in [24], and the other one is our filter. Both designs have 3×3 window size, 8-bit word-length of samples, and are implemented in Altera Cyclone II FPGA IC board (EP2C20F484C7N, 20K logic cells, 234K memory cells, 1.2V power supply, 50MHz clock frequency).

Fig.9 illustrates the quality of pictures produced by the implemented hardware from the 10% noisy image (Lena, 640×480 pixels in size). Clearly, the image produced by our hardware is better than those generated by the existing median filter design [24].

Fig.9: *Hardware results: original Lena image (640×480 pixels) corrupted by 10% noise (top); Image produced by existing median filter hardware [2] (middle); image produced by the proposed filter hardware (bottom)*

Table 6 summarizes the designs in terms of the total number of logic cells (LC) and memory cells (MC) used in the designs, latency, and the power consumption. The power consumption was evaluated based on Quartus II Power Play Analyzer Tool applied to the layout generated by Altera. Based on the number of logic and memory cells used, we conclude that the proposed design is very compact, fast and low power consuming. Although the design was automatically generated, the maximum clock frequency achievable is high.

4. CONCLUSION

This paper presented a novel noise detection technique and hardware architecture for adaptive lowcomplexity 2D-median filter. Unlike other methods, our technique utilizes dynamically computed imagedependent thresholds which prevent loss of image details. Despite of low computational complexity, the proposed noise detection achieves superior quality of results in terms of PSNR and image quality in comparison to related methods. Prototype FPGA implementation of the adaptive median filter for the window sizes of 3x3 has been experimentally compared to existing median filter hardware. As the results show, the proposed design requires less hardware resources and power. Currently we are working on custom VLSI chip implementation of the proposed design.

Table 6: *Comparison on computational complexity*

Design	Design parameters					
	LC	MC	Latency (clocks)	Thermal Power (mWatt)		
				Dynamic	Static	I/O thermal
[24]	733	399	6	20.4	52.3	102.7
ours	638	290	8	18.7	49.4	49.5

References

[1] T.Nodes, and N.Gallager, "Median filters: some modifications and their properties, *IEEE Trans. Acoustics, Speech and Signal Processing*, vol.ASSP-30, no.5. pp.739-746, Oct.1982

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Boston, MA, USA: Addison-Wesley Longman, Publishing Co., Inc., 1992..

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

[4] S.-J. Ko and Y. Lee, "Center weighted median filters and their applications to image enhancement, *IEEE Trans. on Circuits and Systems*, vol. 38, no. 9, pp. 984-993, 1991.

[5] T.Sun, and Y.Nuevo, "Detail-preserving median filters in image processing, *Pattern Recognition Letters*, vol.15, pp.341–347, April 1994.

[6] D.A. F. Florencino and R.W.Schafer, "Decision-based median filter using local signal statistics, Proceedings SPIE Symp. *Visual Communications and Image Processing*, vol.2038, pp.268–275, Sept.1994,.

[7] E.Abreu, M .Lightstone, S.K.Mitra, and K.Arakawa, "A new efficient approach for removal of impulse noise from highly corrupted images, *IEEE Transactions on Image Processing*, vol. 5, pp. 1012-1025, June 1996.

[8] T.Chen, K-K.Ma, L-H.Chen, "Tri-state median filter for image denoising, *Transactions on Image Processing*, vol.8, no. 12, pp. 1834-1838, Dec. 1999.

[9] T.Chen and H.Wu, "Adaptive impulse detection using centerweighted median filters, *IEEE Signal Process. Letters*, vol.8, no.1, pp.1-3, Jan.2001.

[10] S.Zhang and M.Karim, "A new impulse detector for switching median filters, *IEEE Signal Processing Letters*, vol.9, no.11, pp.360-363, Mar.2002.

[11] X.D.Jiang, "Image detail-preserving filter for impulse nose attenuation, *IEE Proceedings Vis. Image Signal Processing.* vol.150, No.3, pp.179-185, June 2003.

[12] I.Andreadis and G.Louverdis, "Real-time adaptive image impulse noise suppression, *IEEE Transactions on Instrumentation and Measurements*, vol.53, no.3, pp.798-806, June 2004.

[13] I.Aizenberg, C.Butakoff, "Effective impulse detector based on rank-order criteria, *IEEE Signal Processing Letters*, vol.11, no.3, pp.363-366, Mar.2004.

[14] W.Luo, "Efficient removal of impulse noise from digital images, *IEEE Transactions on Consumer Electronics. Letters*, vol.52, no.2, 523-527, May 2006.

[15] W.Luo, "An efficient detail preserving approach for removing impulse noise in images, *IEEE Signal Process. Letters*, vol.13, no.7, 413-416, July 2006.

[16] K.S.Srinivasan, and D. Ebenezer, "A fast and efficient decisionbased algorithm for removal of high density impulse noises," *IEEE Signal Process. Letters*, vol.14, no.3, 189-192, Mar 2007.

[17] P-Y.Chen, C-Y.Yuan Lien, "An efficient edge-preserving algorithm for removal of salt-and-pepper noise," *IEEE Signal Processing Letters*, vol.15, pp. 833-836, 2008.

[18] P-Y.Chen, C-Y.Yuan Lien, H-M. Chuang, "A low-cost VLSI implementation for efficient removal of impulse noise," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.18, no.3, pp. 2005-2008, March 2010.

[19] Z.Wang and D.Zhang, "Progressive switching median filter for the removal of impulse noise from highly corrupted images," *IEEE Trans. Circuits and Systems II, Analog and Digital Signal Processing*, vol.46, no.1, pp.78-80, Jan.1999.

[20] R. H. Chan, C.-W. Ho, and M.Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization, *IEEE Transactions on Image Processing*, vol. 14, no. 10, pp. 1479-1485, Oct. 2005.

[21] P.-E.Ng and K.-K. Ma, "A switching median filter with boundary discriminative noise detection for extremely corrupted images," *IEEE Transactions on Image Processing*, vol.15, no.6, pp.1506-1516.

[22] N.I. Petrovic and V.Crnojevic, "Universal impulse noise filter based on genetic programming," *IEEE Transactions on Image Processing*, vol. 17, no.7, pp. 1109-1120, July 2008.

[23] V.G.Moshnyaga, K.Hashimoto, "An Efficient implementation of 1-D median filter," *IEEE Midwest Symp. On Circuits and Systems*, 2009.

[24] C. Chakrabarti: "Sorting network based architecture for median filter," *IEEE Trans. Analog and Digital Signal Processing*, vol.40, no.11, pp.723-727, Nov.1993.

Takeaki Matsubara received the B.E. .degree in Electronic Engineering and Computer Science from Fukuoka University, Japan in 2008. He is now a master student of Graduate School of Electronics Engineering and Computer Science, Fukuoka University, Japan. His current research interests are in the areas of video and image processing and VLSI design.

Vasily G. Moshnyaga received the Computer Engineering Degree with Honors from Technical State University, Sevastopol, USSR in 1980 and Ph.D. in computer engineering from Moscow Aviation Institute in 1986. Till 1992 he was a faculty of Technical University of Moldova, Chisinau, Moldova. From 1992 to 1998 he was a lecture at the Department of Electronics and Communication of Kyoto University, Japan. Since 1998 he has been with Fukuoka University, Japan, where he is currently a Professor at the Department of Electronics Engineering and Computer Science. In 2005-2006 he was a visiting scientist of Computer Science Department, UCLA. His current research interests are in the areas of computer architecture, video processing, VLSI design and design methodologies with a particular emphasis on energyefficient design techniques. He has authored or co-authored over 170 referred journal and conference publications and holds five patents. Dr. Moshnyaga served as Vice-Chair of the IEEE CAS Society, Fukuoka Chapter from 2008 to 2010, Associate Editor of the IEICE Transactions on Fundamentals of Electronics, Communication and Computer Sciences (from 2005 to 2008), and a member of organizing committees of the Asia-Pacific Design Automation Conference, Asia Pacific Conference on Chip Design Languages. He is now a member of IEEE CAS Technical Committee on VLSI and a member of Technical Program Committees of several conferences and symposia including IEEE International Symposium on Circuits and Systems, ACM/IEE International Symposium on Low-Power Electronics and Design, IEEE System on Chip Conference, etc. Dr. Moshnyaga received the Nikkei LSI IP Design Award in 2001, and SCI'2000/ISAS2000 Best Paper Award in 2000. He is a member of IPSJ, IEICE and a senior member of IEEE.

Koji Hashimoto received the B.E and M.E. and Ph.D. Degrees in computer science from Kyushu University, Japan in 1997 and 1999, and 2002, respectively. From 2002 to 2005 he was with Seiko Epson Corp., Japan working as an engineer in Semiconductor Division. Since 2005 he has been with Fukuoka University, Japan, where he is currently an Assistant Professor at the Department of Electronics Engineering and Computer Science. His research interests are in the areas of computer architecture, computer arithmetic, VLSI design, and application specific processors. He has authored or co-authored over 30 referred journal and conference publications and holds three patents

Dr. Hashimoto received the Nikkei LSI IP Design Award in 2001. He is a member of Technical Program Committees of IEEE COOL-Chips, Asia-Pacific Design Automation Conference, etc. He is a member of IEICE and IEEE.