Morphological and Median Adaptive Filters Based on LCBP Rank Filter

Dragana Prokin, *Member*, *IEEE* and Milan Prokin, *Member*, *IEEE*

Abstract — **The presented median and morphological (min and max) filters based on low complexity bit-pipeline (LCBP) rank filter provide reduced complexity of required processing hardware, due to similar pipeline stages and the complete absence of sorting networks in comparison with other solutions. FPGA realization of bit-pipeline median and morphological filter and adaptive bit-pipeline rank filter according to this paper provides significantly higher maximum operating frequency and much smaller used chip resources in comparison with state-of-the-art sorting methods.**

Keywords — **Median filters, morphological filters, adaptive filters, pipeline processing, FPGA implementation.**

I. Introduction

Rank filtering is a nonlinear filtering operation widely used in signal processing, performed by selecting a sample with a specified rank from a one-dimensional (1D) or multidimensional window of samples [1]. Rank filtering of 1D signal is performed by sliding a window of length N input samples over an input signal. The output result (rank sample) of order R from N input samples has R input samples less or equal to the output result, as well as N-R input samples greater or equal to the output result (Fig 1).

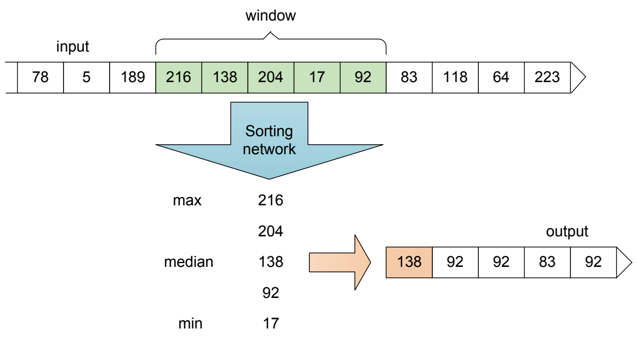


Fig. 1. Rank filtering.

Median and morphological filters (minimum and maximum filters) are the most popular specific cases of rank filters, in which the rank is fixed to R=k=N/2 (integer division), R=0 or R=N-1, of the sorted N window elements respectively. Median filters remove impulse noise without blurring the edges in images. Minimum and maximum filters are used for growing and contracting regions and lines in images during pattern recognition [2], [3].

Algorithms and methods for rank filtering can be classified into two categories: bit-level and word-level. In most cases, only hardware implementations are feasible for real time signal processing applications [4]. The hardware architectures for implementing median and morphological filters are based on different methods: threshold decomposition [5], [6], bit serial [7]-[10], histogram [11], insert-delete approach [12] and sorting network based algorithms [13]-[18].

Most embodiments of median and morphological filters are based on a sorting network. Existing sorting algorithms are based on arranging the elements in ascending or descending order and then selecting the output based on a desired rank of the filter. Sorting networks are very simple devices that only perform compare-exchange operations (Fig. 2). Architectures that use bubble sort approach [16] require N·(2N+1) dual input sorters. This method is highly regular, but the hardware requirements increase in proportion to the window size. The odd-even transposition sorting network [12] is a pipelined structure consisting of N stages with (N-1)/2 compare and swap units in each stage (Fig 3).

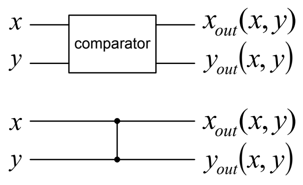


Fig. 2. Compare block.

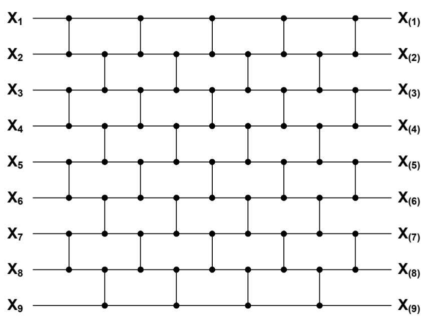


Fig. 3. Odd-even transposition sorting network.

Each unit operates alternately on both odd and even pairs of W-bit window elements. At the output of the last stage, input elements will be sorted in such a manner that the largest element will be at the top and the median at the middle. The propagation delay increases with W. Merge sorting algorithms repeatedly merge the pairs of sorted

This work was partialy supported by the Ministry of Education and Science of the Republic of Serbia under grant no. TR32039 and TR32043.

Corresponding Dragana Prokin is with the School of Electrical and Computer Engineering of Professional Studies, Vojvode Stepe 283, 11000 Belgrade, Serbia (tel: +381-11-395-0035, e-mail: dprokin@viser.edu.rs).

Milan Prokin is with the University of Belgrade — School of Electrical Engineering, King Alexander's Boulevard 73, 11120 Belgrade, Serbia (tel: +381-11-3218-310, e-mail: proka@el.etf.rs).

subset until the entire input set is sorted. Bitonic mergesort [17] is a parallel algorithm for sorting. It is also used for building a sorting network. (Fig. 4).

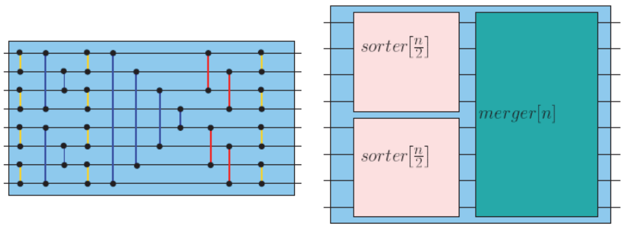


Fig. 4. Merge sorting network.

The fast algorithm RBMSN [18] is based on the minimum-maximum rank range merging of the sorted subset which reduces the number of input elements at each stage of the sorting network. The computational complexity is further reduced by eliminating comparators which do not contribute to computation of the particular desired output rank. The throughput of the algorithm is further optimized by pipelining. Pipelining is achieved by introducing registers for data storage, in order to overlap processing. The throughput of pipelined realization is limited by input/output bandwidth and clock of the system.

Unlike most sorting algorithms, where the number of comparators is data dependent, RBMSN builds a sorting network that is range dependent. The number of elements contending for the output place is reduced after each consecutive stage of sorting, thus a reduction in the number of required comparators is achieved compared to the conventional methods. The optimization in terms of area and speed due to a reduction in the number of comparators also depends on the desired rank of the output. Since the median is the (N+1)/2 largest value, its search by merge sorting method usually requires $3(N^2-1)/8$ comparisons [1], while RBMSN algorithm requires only $N/2\log_2 N$ comparisons.

The purpose of this paper is to describe the advantages of using a LCBP rank filter, presented in [8] and [9] in morphology and adaptive filters. LCBP of N input samples with W bits each is producing a single bit of the output result in each pipeline stage, and accumulating them in order to produce the output result per each clock cycle, independently of the number N of input samples. LCBP works without sorting networks and masking registers, contrary to all state-of-the-art methods.

II. Algorithm Description

The sliding window comprises N unsigned integer input samples X[j], W bits each, such as one-dimensional digital signal samples:

$$X[j] = \sum_{i=0}^{W-1} b_i(j) \cdot 2^i \tag{1}$$

where $b_i \in \{1,0\}$, j = 0,..,N-1.

MSBs of all N input samples X[j] are examined first. The number of bits with a zero value in each input data sample is counted by the N-bit adder of zeros. The result of this adder is compared with the desired rank and if it is greater than the rank, the output of the comparator (the result bit) becomes zero. In the next stage bits (MSB-1) will be examined, and so on until LSBs. After passing through W delaying deskew registers, the result bit will become the MSB of the output result, equal to the rank sample (selected input sample).

Table 1 shows an example where five unsigned 8-bit words (with values of 156, 14, 202, 224 and 68 respectively) are rank ordered. The window size is N = 5, and the rank value is RANK=3. The result Y=156=10011100 is the third smallest among these five numbers.

TABLE 1: AN EXAMPLE OF RANK ORDERING ALGORITHM.

Word (X)	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
10011100	1	0	0	1	1	1	0	0
00001110	0	0	0	0	0	0	0	0
11001010	1	1	1	1	1	1	1	1
11100000	1	1	1	1	1	1	1	1
01000100	0	0	0	0	0	0	0	0
Result (Y)	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**

Steps in the algorithm described in [8], [9] are repeated for every stage, in order to generate the output result:

Step 1: Get W-bit data samples X[j], j=0,..,N-1, and the RANK input.

Step 2: Get the MSBs from k-th stage. Count number of zeros.

If number of zeros >= RANK, then

- Output result bit of the k-th stage, Y[k]=0.
- All other bits in an input data sample will be kept in case that MSB is zero.
- All other bits in an input data sample will be set in case that MSB is one.

Else if number of zeros < RANK, then

- Output result bit of the k-th stage, Y[k]=1.
- All other bits in an input data sample will be kept in case that MSB is one.
- All other bits in an input data sample will be reset in case that MSB is zero.

Step 3: If k > 0, then k = k-1. Repeat Step 2.

Step 4: Output Y[W-1:0].

III. Hardware Implementation

Each control logic block in pipeline stage (Fig. 5) passes through or modifies input sample bits according to the aforementioned algorithm.

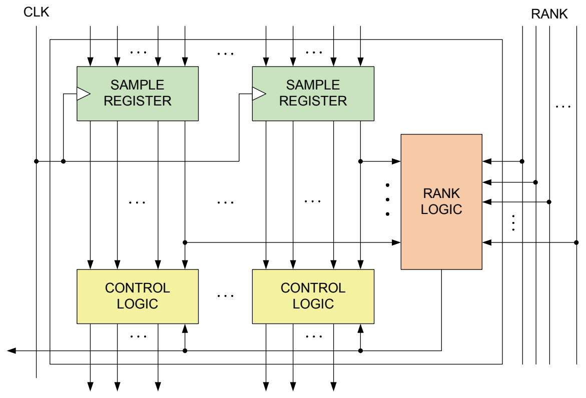
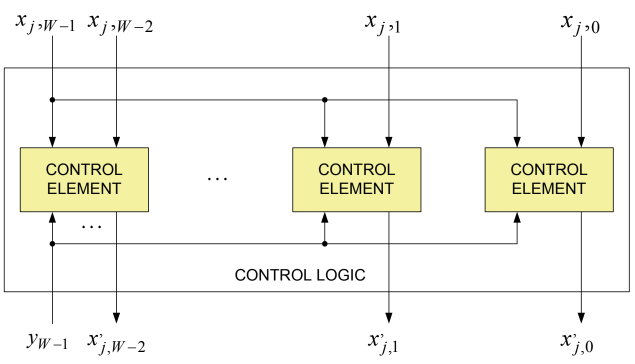


Fig. 5. One pipeline stage.

Each control logic block (Fig 6) consists of control elements (CE), feeding sample registers in the next stage

(Fig. 7). Inputs $x_{j,i}$ are the MSBs of the previous sample register and y_i is the appropriate output result bit. In case that these two bits are identical, the output $x'_{j,i-1}$ is $x_{j,i-1}$. Otherwise, MSB-1 bit of the next sample register will become $x_{j,i}$, i.e., the MSB of the previous sample register.



$x_{j,W-1}$ $x_{j,W-2}$ $x_{j,1}$ $x_{j,0}$
CONTROL ELEMENT
CONTROL ELEMENT
CONTROL ELEMENT
CONTROL LOGIC
y_{W-1} $x'_{j,W-2}$ $x'_{j,1}$ $x'_{j,0}$

$$x'_{j,i-1} = x_{j,i} \cdot \overline{y_i} + x_{j,i-1} \cdot (x_{j,i} + \overline{y_i})$$

Fig. 6. One control logic block.

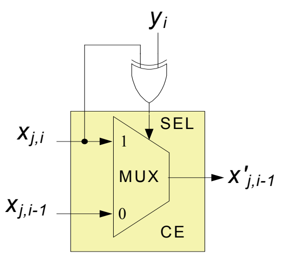
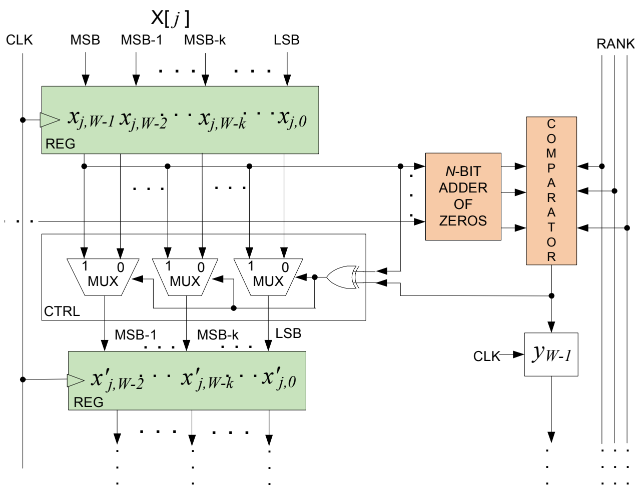


Fig. 7. One control logic element.

The operation of the control logic block (CTRL) is presented using the modification of bits from the previous sample register (REG), comprising input sample $x[j]$ bits $x_{j,i}$, wherein i = W-1,…,0 (Fig. 8). After passing through the control block (CTRL), modified values are stored in the next sample register (REG) as $x'_{j,i}$ wherein i = W-2,…,0.



X[j]
CLK
MSB
MSB-1
MSB-k
LSB
RANK
$x_{j,W-1}\, x_{j,W-2} \cdots x_{j,W-k} \cdots x_{j,0}$
REG
N-BIT ADDER OF ZEROS
COMPARATOR
MUX
MUX
MUX
CTRL
MSB-1
MSB-k
LSB
CLK
y_{W-1}
$x'_{j,W-2} \cdots x'_{j,W-k} \cdots x'_{j,0}$
REG

Fig. 8. The example of first stage (W-1) control block.

The selection of one of multiplexer inputs is provided by the output of XOR circuit, based on MSB of the previous sample register (in a general case $x_{j,i}$ or in this case $x_{j,W-1}$) and the comparator output (in a general case y_i or in this case y_{W-1}). In case that these two bits are identical, the multiplexer output $x'_{j,i-1}$ is the next bit $x_{j,i-1}$ (MSB-1) of the previous sample register (REG). Otherwise, the multiplexer output $x'_{j,i-1}$ is the $x_{j,i}$ (MSB) of the previous sample register (REG).

In the proposed LCBP solution, the rank logic block generating the output result bit of the appropriate weight in each pipeline stage, consists of an N-bit adder and a comparator, providing the least complex solution in FPGA. However, this part can be realized in any other manner, which is more optimal considering the utilization of hardware resources of the particular chip.

After passing through delaying deskew registers in each pipeline stage (Fig. 9), the result bit will become the output result, equal to the rank sample (selected input sample).

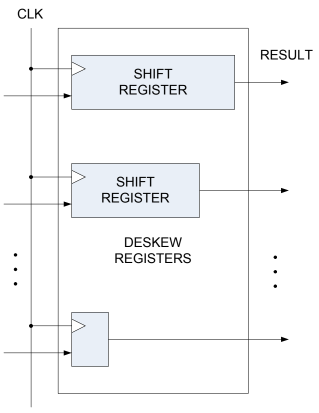


Fig. 9. Deskew registers.

In cases of minimum or maximum filter, the appropriate rank logic block is further simplified to:

Max: $y_i = x_{0,i} + x_{1,i} + x_{2,i} + \ldots + x_{N-1,i}$

Min: $y_i = x_{0,i} \cdot x_{1,i} \cdot x_{2,i} \cdot \ldots \cdot x_{N-1,i}$

LCBP rank/median filter also benefits from the absence of complex bubble sort, selection sort, merge sort, quick sort and odd-even transposition sort networks in comparison with [10]-[18]. The decreased complexity of LCBP rank/median filter also provides simpler routing of signals and avoids the insertion of additional logic cells and logic expanders in signal paths.

IV. LCABP MEDIAN FILTER

An adaptive median filter can be considered as an iterative 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. The basic algorithm of the adaptive median filter described in [17] has four steps:

- Step 1: Initialize the smallest window size W=3 and maximum window size (W_{max}).

- Step 2: Generate minimum value $x_{\min}$, median value x_{med} and maximum value $x_{\max}$ using the selected rank filter (LCBP in our case).
- Step 3: Evaluate terminating conditions

If $x_{\min} < x_{pr} < x_{\max}$, then the pixel x_{pr} is not corrupted by noise and the output value y_{pr} is the value of the original pixel, i.e. $y_{pr}=x_{pr}$. If $x_{\min} < x_{pr} < x_{\max}$ is not satisfied, then the output value y_{pr} is the median of the window, i.e. $y_{pr}=x_{\text{med}}$, and the processing continues.

- Step 4: Increase window size W.

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 window size W has to be increased.

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

Although the adaptive median filter is defined as an iterative filter, it is possible to generate the result in a two-step process using multiple LCBP rank filters with a different number of inputs from 3x3 to $W_{\max}$x$W_{\max}$, operating in parallel. The minimum value $x_{\min}$, median value x_{med} and maximum value $x_{\max}$ are generated in each LCBP rank filter. However, as these LCBPs have different latencies it is necessary to delay outputs of LCBPs with a smaller window size W in order to have all outputs available at the same time. In the second step, one output is selected by the selector using the aforementioned algorithm (Fig. 10).

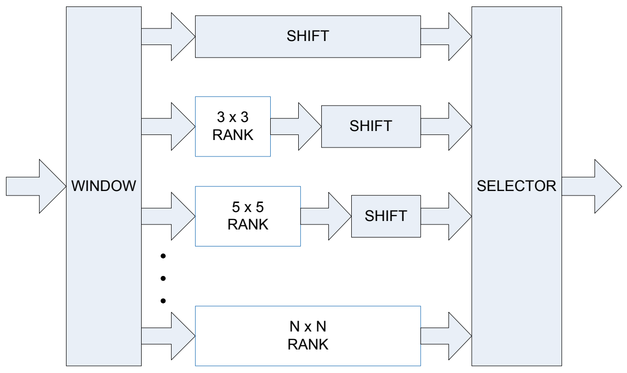


Fig. 10. Hardware implementation of LCABP rank filter.

V. Test results

LCBP filtering algorithm is relatively simple, allowing straightforward hardware implementation. VHDL design of LCBP median, min and max filters have been compiled and simulated using Xilinx ISE Design Suite v11.1 in Xilinx FPGA Virtex II E (Table 2) and Xilinx FPGA Virtex II Pro (Tables 3 and 4). Test results clearly demonstrate a significantly higher maximum operating frequency and much smaller used chip resources in comparison with bit-pipeline filters based on sorting methods [17], [18].

Furthermore, conventional sorting algorithms are not practical for larger window sizes (5x5, 7x7 and bigger) due to the growth of the sort network with the square of the window size, as well as a significant decrease in maximum operating frequency [18].

In addition to five times less resource requirements of LCBP filter for the 3x3 window and seven times less for the 5x5 window than RBMSN (Table 2), the maximum operating frequency of LCBP filter is almost five times higher for the 3x3 window, and more than ten times higher for the 5x5 window. It should also be noted that the maximum operating frequency of LCBP filter is decreased about 2% for the 5x5 window, in comparison with the 3x3 window, while a decrease in the maximum operating frequency of RBMSN the filter is greater than 50%. The number of input / output ports of LCBP filter increases proportionally with the increase in window size, while the required number of input / output ports for RBMSN filter implementation with the 5x5 window is almost doubled in comparison with the 3x3 window, due to the complexity of the method.

Based on Table 3, it can be concluded that standard sorting algorithms used in the implementation of a median filter [17] have resource requirements which are higher on the average 2.5 times (for the 5x5 window) to 6.7 times (for the 9x9 window) in comparison with LCBP median filter. Simultaneously, LCBP filter has a higher operating frequency for the 5x5 window, approximately the same frequency for the 7x7 window and a 1% lower frequency for the 9x9 window in comparison with [17]. LCBP median filter used in the implementation of an adaptive filter, based on the test results shown in Table 4, for all window sizes, has a maximum operating frequency which is 10MHz higher on the average than the methods described in [17].

Table 2: Median, min and max filters in Xilinx FPGA Virtex E.

Window size	*Rank*	**FPGA VIRTEX-E**							
		LCBP rank filter				***RBMSN* [18]**			
		Flip flops	***SLICEs***	***IOB***	**$f_{\max}$ [MHz]**	***Flip flops***	***SLICEs***	***IOB***	**$f_{\max}$ [MHz]**
3x3	median	596	482/12288 (4%)	81/408	326.61	4448	2595/12288 (21%)	89/408	67.57
	min	596	474/12288 (4%)	81/408	326.61	4448	2562/12288 (20%)	89/408	67.57
	max	596	482/12288 (4%)	81/408	326.61	4448	2562/12288 (20%)	89/408	67.57
5x5	median	1172	758/12288 (6%)	209/408	318.40	9344	5697/12288 (46%)	393/408	29.83
	min	1172	750/12288 (6%)	209/408	318.40	9344	5222/12288 (42%)	393/408	29.83
	max	1172	758/12288 (6%)	209/408	318.40	9344	5222/12288 (42%)	393/408	29.83

TABLE 3: MEDIAN FILTERS IN XILINX FPGA VIRTEX II PRO.

Number N of input samples in a window (WxW)	**FPGA VIRTEX II Pro**					
	LCBP median filter		***Method* [17]**			
			Odd-even merge sorting network		***Bitonic sorting network***	
	f_{max}**[MHz]**	***SLICEs***	f_{max}**[MHz]**	***SLICEs***	f_{max}**[MHz]**	***SLICEs***
25 (5x5)	318.	660/23616 (3%)	305	1582/23616 (7%)	305	1706/23616 (7%)
49 (7x7)	316.	924/23616 (4%)	303	4426/23616 (19%)	303	4815/23616 (20%)
81 (9x9)	312	1539/23616 (7%)	302	9719/23616 (41%)	302	10315/23616 (44%)

TABLE 4: ADAPTIVE MEDIAN FILTERS IN XILINX FPGA VIRTEX II PRO

Maximum number N of input samples in a window (W_{max}xW_{max})	**FPGA VIRTEX II Pro**					
	LCABP median filter		***Method* [17]**			
			Odd-even merge sorting network		***Bitonic sorting network***	
	f_{max}**[MHz]**	***Slices***	f_{max}**[MHz]**	***Slices***	f_{max}**[MHz]**	***Slices***
25 (5x5)	315	1460/23616 (6%)	305	2220/23616 (9%)	303	2024/23616 (9%)
49 (7x7)	312	3035/23616 (13%)	303	7297/23616 (31%)	298	6567/23616 (28%)
81 (9x9)	309	5678/23616 (24%)	302	18120/23616 (77%)	298	16395/23616 (70%)

VI. CONCLUSION

In this paper the comparison of FPGA realizations of various rank filtering methods is presented. For realizations, different FPGA platforms are used.

The proposed LCBP method provides not only a simple pipelined architecture, but also a highly regular structure with repeated modules. LCABP requires significantly less chip resources in comparison with the best sorting networks used in state-of-the-art morphological and adaptive filters. The hardware implementation of morphological and median adaptive filters based on LCBP rank filter can easily fit into modern FPGAs, leaving enough resources to implement other processing logic. The maximum clock rate of the designed filter meets the demands of real-time applications [19], [20].

REFERENCES

[1] S. Marshall, *Logic-based Nonlinear Image Processing*. Bellingham, Washington, 2007, pp. 57-71.

[2] T. Chen and H. R. Wu, "Space variant median filters for the restoration of impulse noise corrupted images," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 8, pp. 784-789, Aug. 2001.

[3] D. S. Richards, "VLSI median filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 38, no. 1, pp. 145–153, Jan. 1990.

[4] J. Gil and M. Werman, "Computing 2D min, median and max filters," *IEEE Trans Pattern Anal. Mach. Intell*, vol. 15, no. 5, pp. 504-507, 1993.

[5] J. Fitch, E. Coyle, N. Gallagher C Jr, "Threshold decomposition of multidimensional ranked order operations," *IEEE Trans. Circuits Syst.*, vol. 32, no. 5, pp. 445-450, 1985.

[6] L. W. Chang and S. S. Yu,, "A New Implementation of Generalized Order Statistics Filter by Threshold Decomposition," *IEEE Trans. Signal Proc.*, vol. 40, pp. 3062-3065, 1992.

[7] C. Chang and V. Punam, "A real-time bit-serial rank filter implementation using Xilinx FPGA," in *Proc. SPIE, Real-Time Image Processing*, Jan. 2008, vol. 6811, pp. 68110F (1-8).

[8] D. Prokin and M. Prokin, "Real-time pipelined rank filter," *Int. J. Imaging Science Eng.*, vol. 1, no. 1, pp. 21-26, 2007.

[9] D. Prokin and M. Prokin, "Low hardware complexity pipelined rank filter," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 6, pp. 446-450, 2010.

[10] T. W. Lee, J. H. Lee, and S. B. Choo, "FPGA implementation of a 3x3 window median filter based on a new efficient bit-serial sorting algorithm," in *Proc. 7th Korea-Russia Int. Symp., KORUS*, 2003, pp. 237-242.

[11] S. A. Fahmy, P. Y. K. Cheung, and W. Luk, "Novel FPGA-based implementation of median and weighted median filters for image processing," in *Proc. Int. Conf. Field Programmable Logic and Applications*, Aug. 2005, pp. 142–147.

[12] T. S. Huang, G. J. Yang, G. Y. Tang, "Fast two dimensional median filtering algorithm," *IEEE Trans. Accoust., Speech, Signal Proc.* vol. ASSP-27, pp. 13-18, 1979.

[13] B. K. Kar and D. K. Pradhan, "A new algorithm for order statistic and sorting," *IEEE Trans. Signal Process.*, vol. 41, no. 8, pp. 2688-2694, Aug. 1993.

[14] K. Oflazer, "Design and implementation of a single chip 1-D median filter," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 31, no. 5, pp. 1164–1168, Oct. 1983.

[15] L. W. Chang and J. H. Lin, "A bit-level systolic array for median filter," *IEEE Trans. Signal Process.*, vol. 40, no. 8, pp. 2079-2083, Aug. 1992.

[16] K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel algorithm for general purpose median filtering on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2002, vol. 4, pp. 425-428.

[17] Z. Vasicek and L. Sekanina, "Novel hardware implementation of adaptive median filters" in *Proc. 11th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop,* Apr. 2008, pp. 1-6.

[18] S. M. Meena, and K. Linganagouda, "Rank based merge sorting network architecture for 2D median and morphological filters," in *Proc. Int. Advance Computing Conf*, Mart. 2009, pp. 473–479.

[19] J. Scott, M. Pusateri, and M. U. Mushtaq,; "Comparison of 2D median filter hardware implementations for real-time stereo video," in *Proc. 37th IEEE Applied Imagery Pattern Recognition Workshop*, Oct. 2008, pp. 1–6.

[20] V. R. Vijaykumar, D. Ebenezer, and P. T. Vanathi, "Detail preserving median based filter for impulse noise removal in digital images," in *Proc. 9th Int. Conf. Signal Processing*, Oct. 2008, pp. 793–796.