A Combined VLSI Architecture for Nonlinear Image Processing Filters

Orlando J. Hernandez, Tara Keohane, and Julia Steponanko
Department of Electrical and Computer Engineering
The College of New Jersey
hernande@tcnj.edu

Abstract

In recent years, nonlinear methods and techniques have emerged as intensive research topics in the fields of signal, image, and video processing. This paper describes the combined design of high performance architectures for different filter algorithms and structures used in nonlinear image processing targeted at computer vision. These types of filters include Weighted Order Statistics (WOS), stack, nonlinear mean, Teager, polynomial and rational filters. This architecture is suitable for real time and high quality video imaging tasks and VLSI implementations. These architectures can be used as coprocessor subsystems, integrated with other modules, to form an overall high performance imaging pipe in a system-on-a-chip platform.

1. Introduction

There have been many advances in nonlinear digital image filtering in recent years. More accurate image processing requires the use of nonlinear algorithms because images do not produce linear signals. In the past, the hypotheses of Gaussianity and stationarity were used to produce valid linear models, however, these filters were not capable of removing all noise without distorting the signal. For instance, impulsive noise can be removed from an image using a linear filter; however, edges and small details are blurred in the process [1]. Recent work has delved into advancing the architectures of individual, nonlinear filters in response for this demand for more accurate filtering processes.

Most of the resent work has dealt with the median filter [2]-[11]. This type is the most widely used, robust, and easiest to implement [2]. However, it does not solve all the filtering problems, and is not always efficient. The work presented in [3] consists of four different architectures for the median filter. The implementation discussed in [4] focused on cascaded and recursive stack filters, but proved to be very complex and rarely used. Instead, it was proposed that dynamic domino logic should be used, which resulted in a clocked pipelined architecture. Fibonacci p-codes were found to be a computationally efficient algorithm for general stack filters [2]. A noticeable advantage of using positive Boolean function (PBF) based approaches is that the computations can be done at the bit level, which allows for compact VLSI implementations. Such a PBF pipeline architecture for running order computation was proposed in [5]. [6] focused mainly on the impracticality of designing a stack filter based upon regular threshold decomposition, providing alternative decomposition schemes that require a more reasonable number of binary filters. The architectures presented here also utilized PBFs for implementation. [7] presented stack filters as an extension of rank order filters, presenting designs utilizing minimum, maximum, and median filters. [8] researched the design of weighted order statistic using stack filters as a base. [9] presents the design for a multifunctional filter, one capable of performing numerous rank order filters and computing different variations of erosion and dilation. Some centralized control unit is necessary to determine the type of filtering to be done.

The architectures presented focus on one type of filter and attempt to optimize it for speed and throughput. While [9] has designed a system that implements multiple rank order filters, there is no suggestion of a filter that can implement various types of useful filters in a short amount of time. These considerations are the motivation for the universal filter design.

A more comprehensive image processor can be obtained by combining these architectures into a universal filter. A simple control system can be implemented to determine which type of filter output is desired. This paper proposes one such combination, utilizing a stack filter, weighted order statistic filter, nonlinear mean filter, Teager filter, polynomial filter, and rational filter. One control input determines which individual filter output is the output for the entire universal filter. This filter can be implemented to include many reusable code modules, which reduces the overall size of the filter.

2. Nonlinear image filter algorithms

This section describes the algorithms used for the universal filter. In order to create the architecture, algorithms for the various nonlinear filters used needed to be established. These algorithms were based on a three by three window size, which would scan the image collecting data. All equations are written using a two-input notation to represent the position in the window. Each input is represented by x(horizontal, vertical), where the center input is given by $x(n_1, n_2)$ and the position value increments towards the right and towards the bottom of the window, respectively. Each equation will produce a new value for the center of the window, given by $y(n_1, n_2)$.

The nonlinear image filters used in the universal filter include WOS, stack, nonlinear mean, Teager, polynomial, and rational [1].

2.1. Weighted Order Statistics filter

The Weighted Order Statistics (WOS) filter applies weights to each of the inputs in the window. The center block is assigned a weight of 3, while the four corners are assigned a weight of 1, and the remaining blocks a weight of 2.

The WOS algorithm is given by the following equation:

$$y(n_1, n_2) = \mathrm{MED}\left\{\begin{matrix} w_1 * x(n_1-1, n_2-1) \\ w_2 * x(n_1, n_2-1) \\ w_3 * x(n_1+1, n_2-1) \\ w_4 * x(n_1-1, n_2) \\ w_5 * x(n_1, n_2) \\ w_6 * x(n_1+1, n_2) \\ w_7 * x(n_1-1, n_2+1) \\ w_8 * x(n_1, n_2+1) \\ w_9 * x(n_1+1, n_2+1) \end{matrix}\right\} \qquad (1)$$

where $'w * x'$ means x repeated w times. For example, $5 * 2$ means 2 repeated five times or 2, 2, 2, 2, 2. Essentially what this algorithm does is take the values from the window and assigns a weight to each value. These values are then ordered, or ranked, and the median can be taken, as Eq. (1) indicates.

2.2. Stack filter

The stack filter algorithm is given by the following equation:

$$y(n_1, n_2) = \mathrm{MED}\left\{\begin{matrix} x(n_1-1, n_2-1) \\ x(n_1, n_2-1) \\ x(n_1+1, n_2-1) \\ x(n_1-1, n_2) \\ x(n_1, n_2) \\ x(n_1+1, n_2) \\ x(n_1-1, n_2+1) \\ x(n_1, n_2+1) \\ x(n_1+1, n_2+1) \end{matrix}\right\} \qquad (2)$$

where 'MED' is the function that takes the median value of a group of inputs.

This is the equation for all median filters. The stack filter is a uniquely characterized median filter that utilizes threshold decomposition to turn the input values into a series of binary numbers that are then 'stacked.'

2.3. Nonlinear mean filter

The algorithm for the nonlinear mean filter is based on a basic averaging equation. It is shown below:

$$y(n_1, n_2) = \frac{\begin{matrix} x(n_1-1,n_2-1) + x(n_1,n_2-1) \\ + x(n_1+1,n_2-1) + x(n_1-1,n_2) \\ + x(n_1,n_2) + x(n_1+1,n_2) \\ + x(n_1-1,n_2+1) + x(n_1,n_2+1) \\ + x(n_1+1,n_2+1) \end{matrix}}{9} \qquad (3)$$

2.4. Teager filter

The Teager algorithm for a three by three window size is the following:

$$y(n_1, n_2) = \begin{matrix} 3\,x^2(n_1, n_2) \\ -\frac{1}{2}\left[x(n_1+1, n_2+1)\,x(n_1-1, n_2-1)\right] \\ -\frac{1}{2}\left[x(n_1+1, n_2-1)\,x(n_1-1, n_2+1)\right] \\ -x(n_1+1, n_2)\,x(n_1-1, n_2) \\ -x(n_1, n_2+1)\,x(n_1, n_2-1) \end{matrix} \qquad (4)$$

2.5. Polynomial filter

The polynomial and rational nonlinear filters are two of the more complex filters, requiring a fair amount of computation. The algorithm for the polynomial filter is given by the following equation:

$$y(n_1, n_2) = x(n_1, n_2) + \lambda\, x(n_1, n_2) \tag{5a}$$

where:

$$y(n_1, n_2) = \begin{aligned} &[x(n_1 - 1, n_2) - x(n_1 + 1, n_2)]^2 \\ &\quad [2\, x(n_1, n_2) - x(n_1 - 1, n_2) - x(n_1 + 1, n_2)] \\ &+ [x(n_1, n_2 - 1) - x(n_1, n_2 + 1)]^2 \\ &\quad [2\, x(n_1, n_2) - x(n_1, n_2 - 1) - x(n_1, n_2 + 1)] \end{aligned} \tag{5b}$$

λ is the intensity of enhancement, which for now will be equal to one.

2.6. Rational filter

The rational filter algorithm is shown below:

$$y(n_1, n_2) = \begin{aligned} &x(n_1, n_2) + \lambda \\ &[v_{n1}(n_1, n_2)\, c_{n1}(n_1, n_2) \\ &\quad - v_{n2}(n_1, n_2) - c_{n2}(n_1, n_2)] \end{aligned} \tag{6a}$$

λ is the intensity of enhancement (see above) and:

$$v_{n1}(n_1, n_2) = 2\, x(n_1, n_2) - x(n_1, n_2 - 1) - x(n_1, n_2 + 1) \tag{6b}$$

$$v_{n2}(n_1, n_2) = 2\, x(n_1, n_2) - x(n_1 - 1, n_2) - x(n_1 + 1, n_2) \tag{6c}$$

$$c_{n1}(n_1, n_2) = \frac{g_{n1}(n_1, n_2)}{k\, g_{n1}{}^2(n_1, n_2) + h} \tag{6d}$$

$$c_{n2}(n_1, n_2) = \frac{g_{n2}(n_1, n_2)}{k\, g_{n2}{}^2(n_1, n_2) + h} \tag{6e}$$

$$g_{n1}(n_1, n_2) = [x(n_1, n_2 + 1) - x(n_1, n_2 - 1)]^2 \tag{6f}$$

$$g_{n1}(n_1, n_2) = [x(n_1 + 1, n_2) - x(n_1 - 1, n_2)]^2 \tag{6g}$$

$$k = \frac{1}{2\, g_0} \tag{6h}$$

$$h = \frac{g_0}{2} \tag{6i}$$

where g_0 is the position of resonance.

The above-mentioned algorithms are well suited for VLSI implementation, since they can easily be translated into hardware. It is also possible to combine parts of the algorithms so that less hardware is needed.

3. VLSI architecture

This section presents the architecture of the universal filter. The overall design, shown in Figure 1, consists of memory, registers, the individual filters, a multiplexer, and control lines, as well as a normalizer. The memory contains the data taken from the window scanning the present image. There are nine 8-bit registers in the overall architecture, one for each block value in the window. One window is processed at a time through all the filters. The filters implement the algorithms from the previous section, and then a multiplexer selects the chosen filter type. The normalizer is used for the Teager, polynomial and rational filters to solve the problem of dealing with negative numbers. It will be discussed in more detail later in this section, when the filters that utilize it are discussed.

3.1. Weighted Order Statistics filter

Figure 2 shows the overall layout of the first filter, the weighted order statistic (WOS) filter. There are three parts in the WOS filter that the information goes through: weight adding, sorting, and weighted ordering. The values, from the universal filter registers, first get a weight value assigned to it in relation to its window position. This is comprised of a set of nine adders that places a 2-bit weight value in front of the 8-bit window value, creating a 10-bit number. These nine 10-bit values then move to the registers in the sorting section of the WOS filter.

The newly weighted values are now placed into a second set of registers. These registers are used to store the data as it is being sorted. The sorting section uses eight comparators to sort the values, using only the last eight bits. These comparators compare two numbers at a time, resulting in a high value and a low value. Since there are nine values that need to be compared, the eight comparators are broken into two sets of 4 comparators. The first set compare register 1 through register 8, and the second set compare register 2 through register 9. This

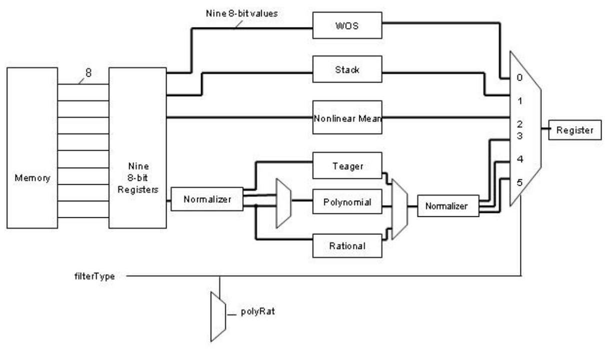

Figure 1. Universal filter block diagram

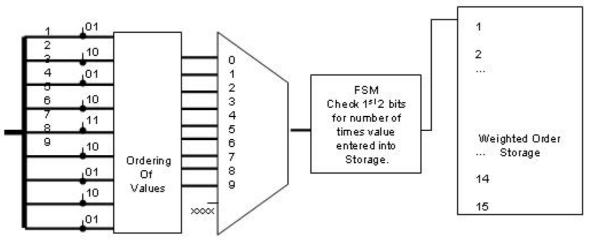
Ordering Of Values
FSM Check 1st 2 bits for number of times value entered into Storage.
Weighted Order ... Storage

Figure 2. Block diagram of the Weighted Order Statistics (WOS) filter

allows an odd number of values to be sorted. Multiplexers determine which value is allowed back into the register, depending on which set of comparators is being used. Basically, what this hardware does is compare the values stored in two adjacent registers, finds the high value and the low value, and swaps the values when necessary. This process continues until the values are sorted from highest to lowest.

The weighted order section is the last section of the WOS filter. The first two bits of the values are checked for their weight values. Recall that the weight of a number represents how many times a number appears in a sequence of numbers. A weight of two means the number will appear twice. For the WOS filter, the weight represents how many times a value will be entered into the shift register. The shift register is made of fifteen 8-bit registers that are wired together so that values can only enter through register 1. When another value goes to register 1, the value in register 1 gets shifted down to register 2. This continues until all fifteen registers have values. Once all the values have been placed in the shift register, the WOS filter is finished, and the final weighted rank can be output, so it can be a selection for the universal filter.

A clock is used for the WOS filter so that the sorting section and the weighted ordering section changes properly. To completely obtain an output from the WOS filter takes 20 clock cycles, which makes this the critical path for the universal filter. The next slowest filter (the rational filter) takes less than half the number of clock cycles. This worst case scenario takes 22.039ns to complete.

3.2. Stack filter

The stack filter implementation required a deviation from the standard threshold decomposition scheme. By accepting an 8-bit input (256 values), the standard stack filter would require 256 separate binary filters, an unacceptable number. Therefore, this stack filter utilizes an alternative threshold decomposition scheme as described in [6] called modular threshold decomposition. This methodology compares the input values to each other to create nine binary outputs instead of 256. For example, the first binary output (x^1) will be a 9-bit number (one bit to represent each of the nine input values) with a one for each input that was greater than or equal to the first input value and a zero otherwise.

Figure 3 shows the final module of the stack filter. The large multiplexer takes in the outputs of the five encoders from the median modules and the encoder takes in the five control lines. These control lines are used to determine which four-bit value is passed through the multiplexer. For instance, if the control lines indicate that one of the values contained five ones, the value from the first median module will be passed. If the control lines indicate that no values contained five ones, it then checks for six, then seven, etc. until it finds the median. The 4-bit output of the multiplexer serve as the control lines for the multiplexer that passes the output value. The implementation of this filter takes 6 clock cycles.

3.3. Nonlinear mean filter

The nonlinear mean filter is the simplest filter in the universal filter, shown in Figure 4. It takes the sum of all the values in the window and divides the sum by the number of values, in this case nine. To ensure accuracy, eight adders are used that take the sum of two numbers. The first adder takes the first two values from the window and adds them together. The next adder uses the sum of the first adder's output and the third value from the window for its two inputs. The third adder uses the sum of the second adder's output and the forth value from the window. This sequence continues until all nine values from the window are added together.

The divider takes the final sum and divides it by nine,

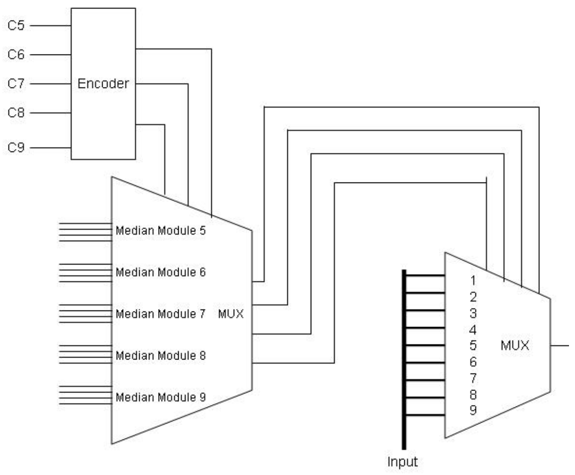

Figure 3. Final module of the stack filter

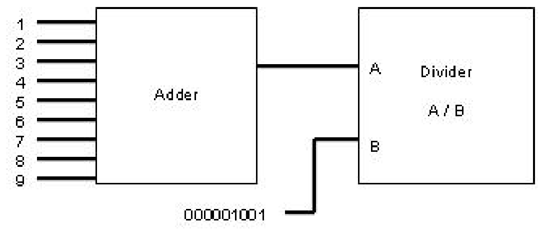

Figure 4. Nonlinear mean filter block diagram

by shifting bits. There are registers for the dividend (the sum of all the window values), divider (nine), remainder, and quotient to store the values while the algorithm shifts. The remainder register is initializes with the dividend. The divider register is subtracted from the remainder register and the result is placed in the remainder register. If the divider went into the dividend, the quotient register is shifted left, setting the rightmost bit to 1. If the divider did not go into the dividend, the quotient register is shifted left, setting the rightmost bit to 0, and the value of the remainder is restored. The divider is then shifted right one bit. This continues until nine divides the final sum. The quotient and remainder are the outputs; here only the quotient will be used for the nonlinear mean output. To get the output from this filter takes 3 clock cycles.

3.4. Teager filter

The Teager filter is one of three filters to utilize the normalizer. The normalizer solves the problem of the Teager algorithm producing a negative number, either as the final result or within the computations. The same holds for the polynomial and rational filters. The normalizer takes the values from the window, which are in the range of 0 to 255, and changes them so they are between -128 and 128 and constitute nine bits. It is assumed that this data is then divided by 128 (simply shifiting the decimal point), leaving all values between -1 and 1.

The hardware of the Teager filter (see Figure 5) implements the algorithm from Eq. (4). There are five multipliers that take the window values and multiply the correct ones together. The center value of the window is squared and multiplied by 3. Values 9 and 1 are multiplied together with ½ to avoid using a divider, which takes longer to execute. The same is true for window values 7 and 3. Finally, values 6 and 4 are multiplied together, and values 8 and 2 are multiplied together. Since four of the multiplications are subtracted from the first multiplication, these four are first added together, and this total is subtracted from the first multiplication. The final result then goes through another normalizer, which reverses the process of the first normalizer, putting the

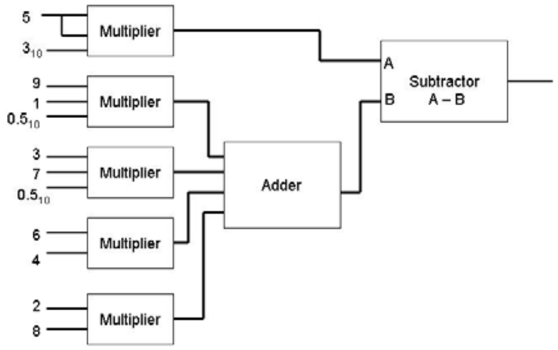

Figure 5. Block diagram of the Teager filter

final result back into the range of 0 to 255. This filter runs 5 clock cycles from beginning to end.

3.5. Polynomial filter

The implementation for the polynomial filter comes directly from Eq. (5). The input signals are first passed through the normalizer. This data is then passed into the filter (see Figure 6). The algorithm possesses numerous stages of multiplication; however, the filter is required to output an 8-bit number. Therefore, the output of each multiplier is rounded to include only the nine most significant bits. Since the filter is assuming that all values are between -1 and 1, this is akin to rounding off the smallest part of the fraction solution.

In this filter, input values 4 and 6 are added together and the sum is subtracted from value five which has been shifted once to the left. This value, which is also coefficient v_{n1} from Eq. (6b) for the rational filter, is then multiplied by the squared difference of values 4 and 6. This same process is repeated for the input values 2 and 8, substituting for values 4 and 6 respectively, as well. This value is also coefficient v_{n2} from Eq. (6c) for the rational filter. These two products are then added together, multiplied by the intensity coefficient λ, and finally added to value 5. The polynomial filter can output a value after 8 clock cycles, making this one of the slower filters.

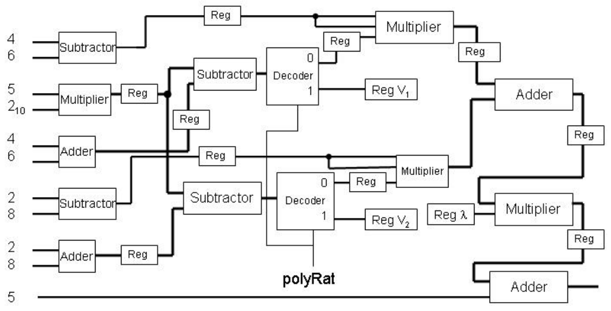

Figure 6. Polynomial filter block diagram

3.6. Rational filter

The rational filter is constructed in a similar manner to the polynomial filter, based simply on Eq. (6). The two coefficients, v_{n1} and v_{n2} are taken directly from the polynomial implementation rather than implementing two identical modules. The rest of the implementation is detailed in Figure 7. Values 8 and 2 and values 6 and 4 are subtracted and both differences are then squared. Each of these values is then squared again and multiplied by the constant k. These two values are the added to the constant h. The first sum is divided by the square of the difference of values 8 and 2 and this quotient is multiplied by the coefficient v_{n1}. The second sum is likewise divided by the square of the difference of values 6 and 4 and multiplied by the coefficient v_{n2}. The two products are added together and added to value 5 to produce the final output. The rational filter, when run in conjunction with the polynomial filter to obtain the coefficients v_{n1} and v_{n2} more efficiently, takes 9 clock cycles to calculate an output.

3.7. Overall implementation of the Universal Filter

The overall implementation is fairly simple, as shown in Figure 1. All of the separate filter modules receive the nine window inputs from an array of nine, eight-bit registers and output into a 6-to-1 multiplexer. The control line, filterType, determines which filter output is passed through the multiplexer. This output replaces value 5 of the input window. The only other control line is polyRat, which, as shown in Figure 6, is an input into the polynomial filter and determines whether the desired output is that of the polynomial filter or the rational filter. The timing for these additional components has been taken into consideration for each of the individual filters discussed above. Therefore, no additional clock cycles

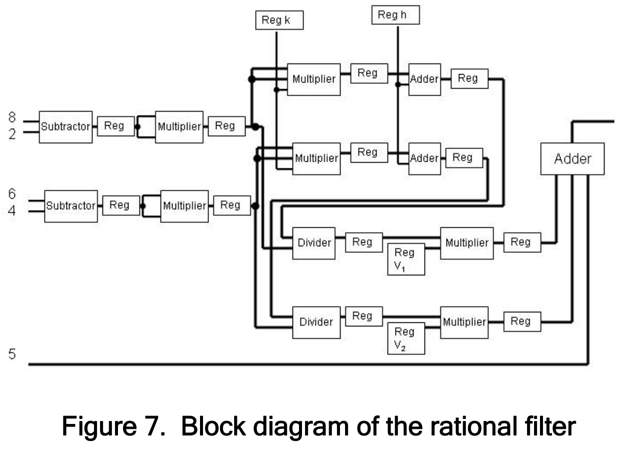

Figure 7. Block diagram of the rational filter

need to added to the timing listed in the preceding sections.

4. Conclusions

This paper proposes a universal filter architecture. It provides a multi-functional image processing solution for applications that require the output of nonlinear filters, such as edge preservation smoothing, noise filtering, and image segmentation. The system's maximum frequency, as determined by the slowest filter, is 45.37Mhz. In the future, more nonlinear filters can be added to the universal filter architecture to make it an even more comprehensive image processing tool. The existing filter modules can also be streamlined as advancements are made in individual filter architectures and as more shared modules are created. This universal filter can easily be modified as nonlinear filtering methods continue to be developed and improved.

5. References

[1] S. K. Mitra and Giovanni L. Sicuranza, eds. *Nonlinear Image Processing,* Academic Press, San Diego, 2001.

[2] A. M. Grigoryan, "Mixed Median Filters and Their Properties," *Proceedings of SPIE Nonlinear Image Processing VIII,* vol. 3026, February 1997, pp. 8-20.

[3] L. Breveglieri and V. Piuri, "Digital Median Filters," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology,* vol. 31, no. 3, July 2002, pp. 191-206.

[4] K. Egiazarian, O. Vainio, and J. T. Astola, "Implementation of Cascaded and Recursive Stack Filters," *Circuit Systems and Signal Processing,* vol. 15, no. 1, 1996, pp. 93-111.

[5] C. T. Chen, L. G. Chen, and J. H. Hsaio, "A Bit-Level Pipelined VLSI Architecture for the Running Order Algorithm," *IEEE Transactions on Signal Processing,* vol. 45, no. 8, August 1997, pp. 2140-2144.

[6] M. J. Avedillo, J. M. Quintana, H. El Alami, and A. Jimenez-Calderon, "A Practical Parallel Architecture for Stack Filters," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology,* vol. 38, no. 2, September 2004, pp. 91-100.

[7] A. Hisat and O. Hasan, "Bit-Serial Architecture for Rank Order and Stack Filters," *Integration, The VLSI Journal,* vol. 36, pp. 3-12, 2003.

[8] C. Chakrabarti and L. E. Lucke, "VLSI Architecture for Weighted Order Statistic (WOS) Filters," *Signal Processing,* vol. 80, 2000, pp. 1419-1433.

[9] A. Gasteratos and I. Andreadis, "Nonlinear Image Processing in Hardware," *Pattern Recognition,* vol. 33, 2000, pp. 1013-1021.

[10] D. Z. Gevorkian, K. Egiazarian, S. Agaian, J. T. Astola, and O. Vainio, "Parallel Algorithms and VLSI Architectures for Stack Filtering Using Fibonacci P-Codes," *IEEE Transactions on Signal Processing,* vol. 43, no. 1, January 1995, pp. 286-295.

[11] A. Gasteratos, I. Andreadis, and P. Tsalides, "Realization of Rank Order Filters Based on Majority Gate," *Pattern Recognition,* vol. 30, Issue 9, September 1997, pp. 1571-1576