FPGA Implementation of a Nonlinear Two Dimensional Fuzzy Filter

Justin G. R. Delva [*] *, Ali M. Reza* [*] *, and Robert D. Turney* [+]

[+] CORE Solutions Group, Xilinx
San Jose, CA 95124-3450, USA

[*] Department of Electrical Engineering and Computer Science, UWM
Milwaukee, Wisconsin 53201-0784, USA

ABSTRACT

Nonlinear filtering has found many practical applications in digital signal and image processing. The computation complexity of these filtering algorithms make them difficult for real-time hardware implementation. One of these nonlinear filters, which is based on fuzzy classification of each pixel to subgroups of its neighboring pixels, is considered here for hardware implementation. The criteria of this filter are based on the local context which form the basis of the fuzzy rule. The filtering algorithm is slightly modified for implementation into a Xilinx Virtex series of FPGA for real-time processing of image sequences. Implementation details and recommendations for further improvement are discussed. Result of a simulation example from the proposed hardware implementation is also presented.

I. Introduction

Noise filtering is an important part of processing and restoring a real image sequence[1]. The noise statistics are usually unknown and differs from one application to another. The more common approach in modeling the noise is to assume that it has Gaussian distribution in addition to some low percentage of additive impulsive noise (pepper and salt noise)[2]. The filtering involves the removal or reduction of impulsive noise along with the Gaussian noise while preserving or enhancing edges[3]. The edges give the image the appearance depth and sharpness. A loss of edges makes the image appear blurred or unfocused. However, noise smoothing and edge enhancement are traditionally conflicting tasks. Since most noise filtering behave like a low-pass filter, the blurring of edges and loss of detail seems a natural consequence. Techniques to remedy this inherent conflict often encompass generation of new noise due to enhancement[3].

A better solution to a seemingly blind approach to noise reduction is to process the image by its individual pixels based upon the trends or appearance of its immediate neighbors. These trends can be defined as statistical characteristics of the neighboring pixels or they can be based on a fuzzy classification of each pixel to subgroups of its neighboring pixels[4]. This approach to filtering has proven to be a better solution in dealing with the problem of preserving edges[5]. A pixel that is near an edge will ideally be combined with the neighboring pixels that lie on the same side of the edge. This requires some form of morphological classification of the local pixels. The method proposed in [5] is based on applying different sub-grouping of pixels within a 2-D moving window. First the mean of each sub-group is estimated based on a robust estimator; i.e. median. Then the center pixel is classified to the sub-groups by using a fuzzy classifier[6]. This algorithm can utilize different morphological structure[6] to capture different patterns and edge characteristics in an image.

In this paper a nonlinear fuzzy filter, originally introduced in [5], is proposed for hardware implementation. The proposed fuzzy filter is tailored for implementation into a Xilinx Virtex series of FPGA for real-time image sequence (video) restoration. The proposed implementation is developed only for the most likely morphology for grouping the neighboring pixels. Extension to other morphological structure is straightforward.

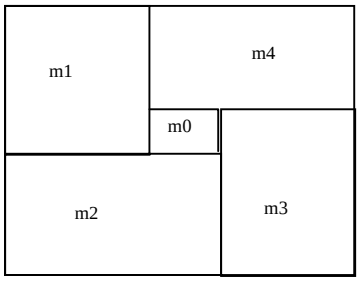

Figure 1: The most likely morphological structure that can be used for all image classes.

II. Nonlinear Fuzzy Filter

The filter proposed in [5] is based on morphological operators introduced in [6]. In this approach, morphologies of square odd sized windows are used. The idea is to find the best geometrical structure for dividing the moving window into four subgroups such that each subgroup is almost totally inside a homogenous area of the image. The best structure depends on the local structure of the edges inside the window. Although the morphology of the best structure for subgroups vary from pixel to pixel, there are only limited number of morphologies that are statistically more likely than any other structures. Based on extensive simulations, it is shown that the most likely structure in all images is as shown in Figure 1. For other likely structures and further discussions, the readers are referred to [5].

The overall procedure is as follows: Different morphological structures are applied to each pixel. In each case fuzzy memberships of the center pixel with respect to the four subgroups are calculated. The morphology with the widest variation among its subgroup fuzzy memberships is chosen as the best fit to the local characteristics of the image. Then based on the available fuzzy memberships and the robust estimates of the means of the subgroups for the selected morphology, the center

pixel is substituted (the filter output is calculated) by the fuzzy average of the subgroups means[5]. This algorithm is well suited for multi processor hardware implementation or FPGA implementation as discussed in the following.

In this paper, implementation of only a single morphological structure is discussed. Expansion to other morphologies is straightforward. The selected window that is used here is the most common morphology found in [5] and is shown in Figure 1. By moving the proposed window, one pixel at a time, over the entire image, the fuzzy classification for each individual pixel is evaluated. At each location, the median of each subgroup is calculated as an estimate of the mean of that particular subgroup. Then based on the value of the center pixel, the fuzzy membership of that pixel with respect to each subgroup is calculated. Finally the filter output corresponding to the center pixel is calculated based on fuzzy averaging of the subgroup medians.

Let m_i represent the median of the *i*-th subgroup, where *i* is taking the values of 1,2,3, and 4. If the center pixel is represented by m_0 , then the membership of the center pixel with respect to the *i*-th subgroup is calculated by

$$w_i = \exp(-(m_0 - m_i)^2 / (2\sigma^2)) + \varepsilon \qquad (1)$$

where σ^2 is the user defined positive constant and ε is a small positive number. The filter output is calculated by using a weighted average of the subgroups means. The weights used in this case are the fuzzy membership functions obtained from Equation (1). The resultant equation for the filter output at each pixel is

$$y = \frac{\sum_i w_i m_i}{\sum_i w_i} \qquad (2)$$

In both Equations (1) and (2) the subscript *i* is changing from 1 to 4.

III. FPGA Implementation

Hardware implementation of the nonlinear fuzzy filtering algorithm is based on the present day FPGA technology. In this implementation, the objective is to demonstrate how a very complex image processing algorithm can be implemented for real-time image sequence (video) processing.

The Virtex by Xilinx was used as the final target of the design. Figure 2 shows the block diagram of the design. Not shown in Figure 2 is a line-buffering scheme used to collect the five lines necessary for the window of Figure 1. *Hline1*, *Hline2*, *Hline3* and *Hline4* are the four twelve bit outputs of the line buffer. These *Hlines* are then fed into a Mask buffer that dissects the 5x5 window into the 4 sub-windows where each contains six pixels. After the Mask buffer, the median calculation is performed on all four sets of six pixel values. This produces the four medians m_1, m_2, m_3 and m_4. The medians and the m_0 point are then used for calculating the membership functions in the Fuzzy Calc section.

The last two stages of the block diagram, consists of the summing of the weighted values, followed by their division by the sum of the weights.

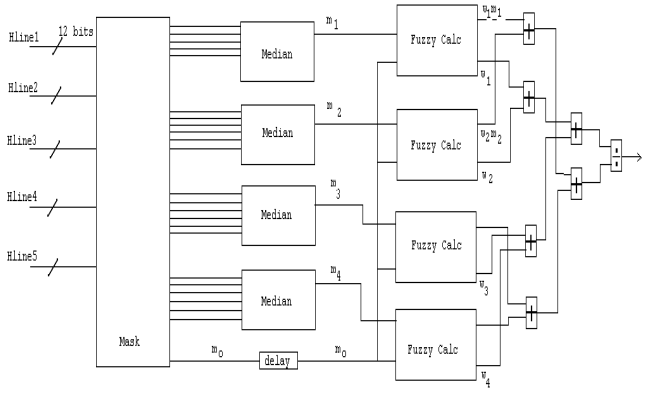

Figure 2: Fuzzy Filter chip layout

Robust Estimator

To perform the median calculation, the pixels in each block must be first sorted in increasing order. After sorting, the median of the six pixels in each block is given by

$$m_0 = median\ (f) = \frac{f(3) + f(4)}{2} \qquad (3)$$

where ***f*** is the sorted sequence of pixels. A merge-sort network [8] is used to sort each block. Merge-sort networks are highly connected structures that are able to sort a sequence S={s_1, s_2, s_3, …, s_n} where *n* is a power of 2. A complete example is illustrated in Figure 3 for the sequence S={8, 4, 7, 2, 1, 5, 6, 3}.

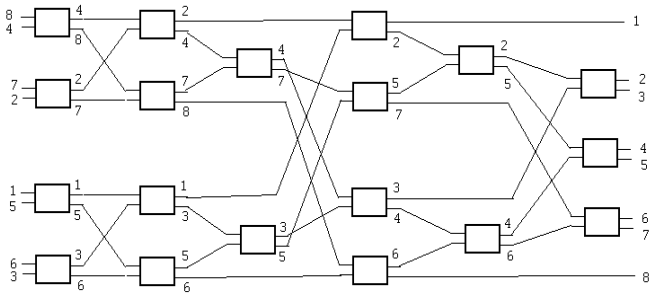

Figure 3: Sorting network for sequence of size eight.

In Figure 3, each cell of the merge-sort network consists of a comparator that outputs the largest and the smallest of the two input values as shown in figure 4.

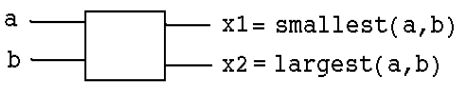

Figure 4: A merge-sort cell

Finding the median from the outputs of the merge-sort network is simply a matter of finding the average of the fourth and fifth output values. For the proposed fuzzy filter, the merge-sort network has to be modified for the six pixel-values (not eight) that need to be sorted. A simple solution is to always assign to zeros the first two values of the pre-sorted sequence. Thus at the

output of the merge-sort network, the inserted zeros will always be the first two output values for any sequence; consequently, the two values needed to compute the median will now be the 5'th and 6'th output. A modified merge-sort network is in order and its architecture is shown in Figure 5.

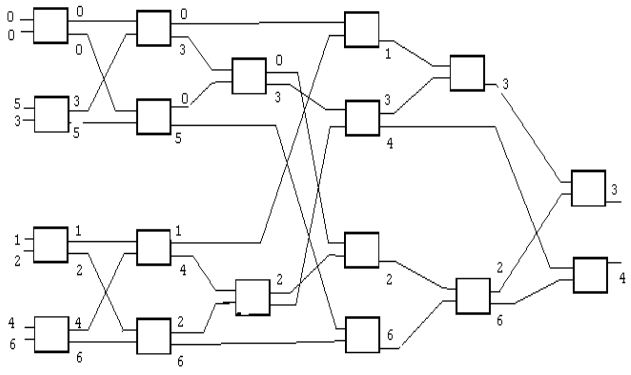

Figure 5: Modified sorting network

The new network is a trimmed version of the full scale network of Figure 3. Further simplification of this network can be envisioned, starting from the comparator whose inputs are hardcoded to zeros. Three cells constitute the building block for the new sorting network: the comparator of Figure 4, the min and max cells shown respectively in Figure 6 and Figure 7.

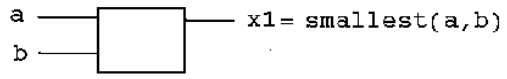

Figure 6: min comparator

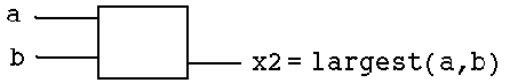

Figure 7: max comparator

Fuzzy Calculation

After finding the medians of each of the sub-windows of Figure 1, the calculation of the four sum terms in both the numerator and the denominator of Equation (2) is executed in parallel in each sub-window. To generate the summing terms in the numerator and denominator of Equation (2), the w_i values must be computed first using Equation (1), then calculation of the four term follows. Figure 8 shows the datapath for the generation of $w_i m_i$ and w_i. A subtractor is used, followed by a look-up-table for the computation of the **exp(.)** function. All components used

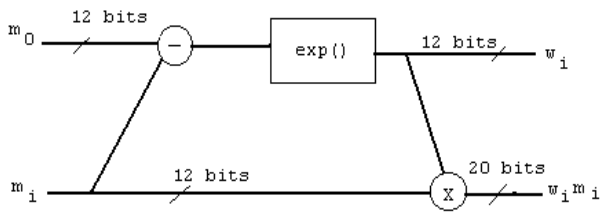

Figure 8: datapth to compute $w_i m_I$ and w_i.

in the design are readily available in Xilinx's Coregen. This application contains important building blocks for DSP designers. The **exp(.)** block is a look-up-table (ROM) that uses as addressing the difference $m_0 - m_i$. The contents of the look-up-table are appropriately scaled for generating good precision. Redundancy in the look-up-table is one drawback in using exponential functions. The redundant terms occur most often for high pixel values. A simple approach to this problem is to first determine the threshold where the redundancy becomes costly, then to use a ROM for values below the threshold (since most values will be below threshold). For values after the threshold, a logic scheme can be used to generate the few remaining terms.

Concerns may arise regarding the size of the engine in Figure 8. The four Fuzz Calc blocks used in Figure 2 could be replaced by a single block that would be time-shared. This scheme would reduce the number of logic cells but falls short of achieving the minimum throughput for video. Another approach is to use Distributed Arithmetic [9] [10] which may use less logic cells.

After the summing terms of Equation (2) are generated, the next obvious step is to sum the values in the numerator and then the denominator. Afterwards, a divider is used to give the final output. The output is trimmed to twelve bits by removing the binary decimal places.

Design Performance

The system can easily operate at a 66 Mhz clock enabling 1024x1024 60 Frames/s operation. System clock rates of 80 Mhz and 100 Mhz can also be achieved for more aggressive system bandwidth requirements. Hardware resources to perform the data path of Figure 2 are approximately 5048 Logic Cells allowing for use of a Virtex family XCV300 by Xilinx.

IV. Simulation

The type of noise used in this simulation is a combination of white Gaussian noise along with impulsive noise known as salt and pepper noise. Images were tested with equal amounts of salt noise (gray level white) and pepper noise (gray level black). The locations of these impulsive noises were randomly selected.

In Figure 9, the Lena image is contaminated with white Gaussian noise to achieve SNR=10db along with 20% salt and pepper noise. The processed image is represented in Figure 10 for which the parameter σ^2 is set equal to 16 and the moving window is 5x5. Careful analysis of the fuzzy filter and further simulations reveal that the window size of 5x5 is performing very well for most general images. Similarly, it is found that the value of 16 for parameter σ^2 in Equation (1) is producing good results.

V. Conclusion

In this work it is shown that present day FPGA technology can be fully explored for implementation of more complex digital image processing algorithms when real-time video processing is necessary. This fact is demonstrated by realization of a very complex nonlinear fuzzy filtering algorithm for a real-time video processing. The fuzzy rule based enhancement is an attractive solution to removing noise while preserving the integrity of edges as much as possible. One past drawback of this type of algorithm was that they required extensive computation which most DSP processors can not easily perform.. With FPGA technology the advantage of highly parallel and pipelined

structures enable image-processing function without the computational overhead found in DSP processors.

Figure 9: Lena image contaminated with Gaussian noise to achieve SNR of 10 and added impulsive (pepper and salt noise) noise of 20%.

Figure 10: Processed image via the proposed hardware implementation.

Bibliography

[1] Anil K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, NJ, 1989.

[2] R. Haddad and T, Rarsons *Digital Signal Processing, Theory, Applications and Hardware*, Computer Science Press, NY, 1991.

[3] D. Marr and E Hildreth, Theory of edge detection. *Proceedings of the Royal Society*, B 207:187-217,1980.

[4] H. J. Zimmerman and P. Zysno, "*Quantifying vagueness in decision models*," *European J. Operational Res*., vol. 22, pp. 148-158, 1985.

[5] Maged S. Riad, "*Image Processing Using Fuzzy Clustering Algorithms*," M.S. Thesis, Electrical Engineering, University of Wisconsin-Milwaukee, May 1995.

[6] R. Krishnapuram and M. Keller, "A Possibilistic Approach to Clustering," *IEEE Trans. on Fuzzy Sys*., vol. 1, No. 2, pp. 98-110, 1993.

[7] J. Serra and L. Vincent, "*An overview of morphological filtering*," Circuit, Systems and Signal Processing, vol. 11, pp. 47-108, 1992.

[8] Selim G. Aki, The Design and Analysis of parallel Algorithms, Prentice Hall 1989

[9] Les Mintzer, "*Large FFT's in a Single FPGA*," ICSPAT Conference, pp. 895-899, 1996.

[10] Xilinx Application Note, "*The role of distributed Arithmetic in FPGA Based Signal Processing*," 1996.