Hardware Implementation for Design of Modified Adaptive Median Filter for Image Processing

Ch.Ravi Kumar
Research Scholar ,Bharath University,Chennai,India
Asst. Professor, Prakasam Engg. College, Kandukur, Prakasam Dt. Andhrapradesh

S.K. Srivathsa
Senior Professor, St. Joseph College of Engg Chennai

Abstract
Digital image processing is an ever elaborating and dynamic area with applications reaching out into our everyday life such as medicine, space exploration, surveillance, authentication, automated industry inspection and many more areas. Applications such as these involve different processes like image enhancement and object detection. Median filtering is a powerful instrument used in image processing. The traditional median filtering algorithm, without any modifications gives good results. There are many variations to the classical algorithm, aimed at reducing computational cost or to achieve additional properties. Median filters are used mainly to remove salt-and pepper noise. The filter logic is implemented on a novel reconfigurable fabric. In this paper look into a Efficient architecture for non-linear modified Adaptive median filter implementation is presented. Then Adaptive Median Filter solves the dual purpose of removing the impulse noise from the image and reducing distortion in the image and the classical adaptive median filter has some deficiencies: the filtered images remain the positive impulse noise in the black background and the negative impulse noise in the white background. To solve the above questions, a modified scheme is proposed. The practical results show the effectiveness of our improvements allowing real-time processing and a minimum use of resources.

1. Introduction

Image Processing is an ever expanding and dynamic area with applications impacting our everyday life in such diverse areas as medicine, space exploration, surveillance, and authentication. In real-time image transmission, images are mostly affected by additive noise (like Gaussian noise) or impulse noise (e.g. salt and pepper noise). The received images should be enhanced in such a way that the Mean Square Error (MSE) of the received image with reference to the original image is minimized. The image enhancement requires a series of filtering operations on the received image. These filters are serially connected since image processing design is divided into a number of processing stages. Implementing such filters on a general purpose computer is straightforward, but not very time efficient due to constraints on processor speed and available memory.

The run-time efficiency of filters can be improved through the use of specially designed Application Specific Integrated Circuits (ASICs) and Digital Signal Processors (DSPs). Fast execution times for complex computations can be achieved by optimizing the circuits for the specific application

System on Chip (SoC) platforms based on ASICs and DSPs can exploit the parallelism and pipelining algorithms to allow for greatly reduced execution times. However, these designs result in a fixed hardware architecture and circuitry, and therefore cannot implement complex systems in an efficient and flexible manner. These limitations can be overcome by leveraging recent advances in Reconfigurable SoC's based on Field Programmable Gate Arrays (FPGAs). The latest version of these reconfigurable systems introduces the concept of 'Dynamic Run-time Partial Reconfiguration,' where only a small portion of the circuitry is modified at run-time while the system remains functioning. Implementing image processing algorithms on reconfigurable hardware allows for the dynamic selection of the filter depending on the characteristics of the received image. This improves the image quality for a wide range of images, while simplifying the debugging and verification process.

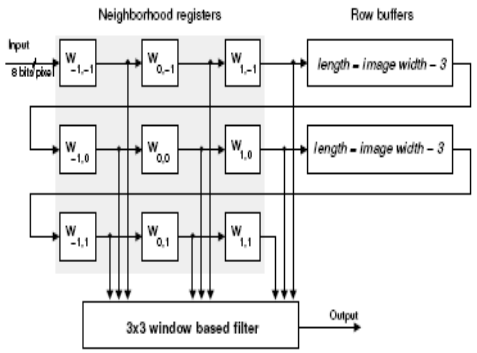

Fig 1.Median filter

Manuscript received March 5, 2010
Manuscript revised March 20, 2010

2. Modified Adaptive Median Filter

The Modified Adaptive Median Filter is designed to eliminate the problems faced with the standard median filter. The basic difference between the two filters is that, in the Adaptive Median Filter, the size of the window surrounding each pixel is variable. This variation depends on the median of the pixels in the present window. If the median value is an impulse, then the size of the window is expanded. Otherwise, further processing is done on the part of the image within the current window specifications. 'Processing' the image basically entails the following: The center pixel of the window is evaluated to verify whether it is an impulse or not. If it is an impulse, then the new value of that pixel in the filtered image will be the median value of the pixels in that window. If, however, the center pixel is not an impulse, then the value of the center pixel is retained in the filtered image. Thus, unless the pixel being considered is an impulse, the gray-scale value of the pixel in the filtered image is the same as that of the input image.

Very diverse FPGA-based custom-computing boards are appearing in the market. These boards possess different interfaces for their communication with the host. But in general, boards devoted to real-time image processing have a USB interface, because it gives them the necessary speed to work as coprocessors. Also, USB bus has a growing popularity due to its interesting properties.

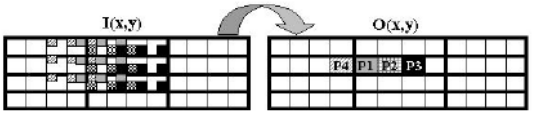

Fig 2. simultaneous computation of output pixel

The fact that we have a 32-bit data bus has a very large influence in the necessary hardware architecture for implementing image processing operations, because it causes that in each read/write operation we obtain/send four image pixels (supposing 8-bit pixels). We have gained benefit from this situation replicating the functional units in order to apply the median filter simultaneously on four pixel neighbourhoods. In this way we take advantage of the inherent neighbourhood parallelism, and we accelerate the operation four times. Figure 2 presents the approach followed for the simultaneous computation of these four output pixels.

Images are divided in pixels (squares) that are grouped in 32-bit words (4 pixels). The value of each output pixel O(x,y) is computed using the 9 pixels of the image I that are inside the 3x3 mask with centre in I(x,y). Each mask application has been represented with a different texture. Note that the pixel P4 of the previous word is computed and not that of the current word. In this way, it is only necessary to read six words in the input image instead of nine, reducing the number of read operations, and therefore increasing the performance. Pipelining this approach using two stages it is possible to get an architecture that writes four pixels (one word) in the output image in each clock cycle, only reading three input image words by cycle

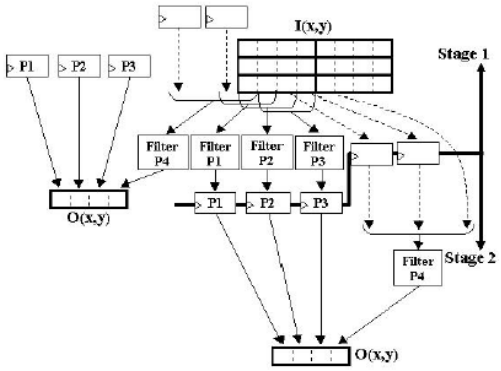

Fig 3. Pipelining approach using two stages

3. Moving Window Architecture

In order to implement a moving window system in VHDL, a design was devised that took advantage of certain features of FPGAs. FPGAs generally handle flip - flops quite easily, but instantiation of memory on chip is more difficult. Still, compared with the other option, off-chip memory, the choice using on-chip memory was clear. It was determined that the output of the architecture should be vectors for pixels in the window, along with a data valid signal, which is used to inform an algorithm using the window generation unit as to when the data is ready for processing. Since it was deemed necessary to achieve maximum performance in a relatively small space, FIFO Units specific to the target FPGA were used. Importantly though, to the algorithms using the window generation architecture, the output of the window generation units is exactly the same. This useful feature allows algorithm interchangeability between the two architectures, which helped significantly, cut down algorithm development time. A window size was chosen because it was small enough to be easily fit onto the target FPGAs, and is considered large enough to be effective for most commonly used image sizes. With larger window sizes, more FIFOs and flip -flops must be used, which increases the FPGA resources used significantly. Figure 1, 2 shows a graphic representation of the FIFO and flip flop architecture used for this design for a given output pixel window.

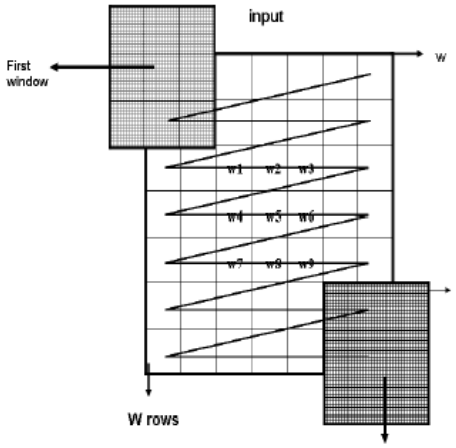

Fig.4 Moving Window Architecture

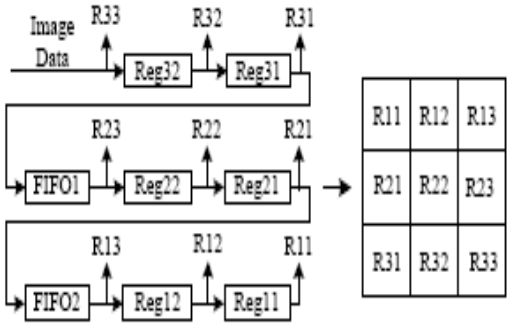

Fig. 5 Reading Pixels from Window.

4. Parallel Sorting strategy:

To make a fair comparison of the parallel sorting strategy against wave sorter strategy in terms of the total number of required steps to sort an array, it is necessary to consider the steps used to read data from memory and the steps required to store the sorted data back to memory. The proposed approach is based on the same structure of the registers array used in the wave sorter strategy. With this kind of array, data can be stored in the array by sending a datum to the first register and later, when the second datum is sent to the first register, the value on the first array is shifted to the second register. Thus, for every datum sent to the array to be stored, values in registers are shifted to their respective adjacent registers. This process requires *n* steps. The same number of steps is required to take data out from the array. This approach allows storing a new set of data in the array while the previous set is being sent back into the memory. As mentioned in section 2, suffix sorting might imply more than one sorting iterations. If *k* sorts are required, then the parallel sorting requires to ((*n+n/2) * k + n)* to sort an array of *n* data. Thus total number of steps required can be obtained by the following equation:

$$f_{steps}^{PS}(n,k) = n\left(\frac{3}{2}k+1\right)$$

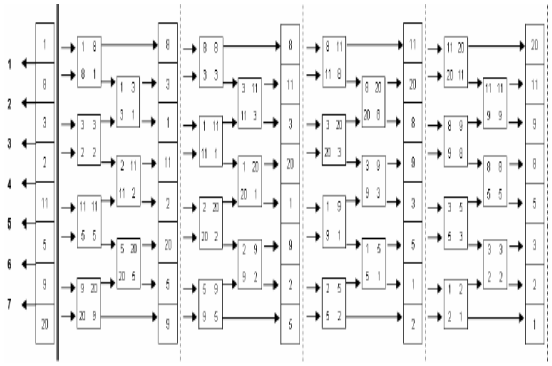

Fig 6.Parallel sorting with two levels of comparators performance

The parallel strategy leads to a significant reduction compared to the wave sorter approach. Furthermore, in additional sorts the necessary number of steps for sorting is equal to the number of characters in the biggest group of identical characters divided by 2 (remember that an additional sorting is implied if groups of identical adjacent characters appear in the array). This implies that in practice, it is possible to reduce more than the number of steps to solve the suffix problem.

5. Implementation and Testing:

The modified adaptive filter works on a rectangular region $\mathbf{S_{xy}}$. The modified adaptive median filter changes the size of $\mathbf{S_{xy}}$ during the filtering operation depending on certain criteria as listed below. The output of the filter is a single value which the replaces the current pixel value at (x, y), the point on which $\mathbf{S_{xy}}$ is centered at the time. The following notation is adapted from the book and is reintroduced here:

$\mathbf{Z_{min}}$ = Minimum gray level value in $\mathbf{S_{xy}}$.

$\mathbf{Z_{max}}$ = Maximum gray level value in $\mathbf{S_{xy}}$

$\mathbf{Z_{med}}$ = Median of gray levels in $\mathbf{S_{xy}}$

$\mathbf{Z_{xy}}$ = gray level at coordinates (x, y)

$\mathbf{S_{max}}$ = Maximum allowed size of $\mathbf{S_{xy}}$

The adaptive median filter works in two levels denoted Level A and Level B as follows:

Level A: A1= $\mathbf{Z_{med} - Z_{min}}$

A2= $\mathbf{Z_{med} - Z_{max}}$

If A1 > 0 AND A2 < 0, Go to level B

Else increase the window size

If window size <= $\mathbf{S_{max}}$ repeat level A

Else output $\mathbf{Z_{xy}}$.

Level B: B1 = $\mathbf{Z_{xy} - Z_{min}}$

B2 = $\mathbf{Z_{xy} - Z_{min}}$

If B1 > 0 And B2 < 0 output Z_{xy}

Else output Z_{med}.

The modified adaptive medianfilter

There are some deficiencies in the above algorithm. It is known that the condition $\mathbf{Z_{min} < Z_{med} < Z_{max}}$ is always satisfied. When $\mathbf{Z_{min} < Z_{med} < Z_{max}}$ go to level B. Otherwise, when $\mathbf{Z_{min} = Z_{min} < Z_{max}}$, $\mathbf{Z_{max=} Z_{min} < Z_{med}}$. it is kept in level A and the window size is increased . When $\mathbf{S_{max}}$ is reached, the algorithm outputs the value of $\mathbf{Z_{xy}}$

Negative impulse appears as black point (pepper noise) in an image and positive impulse appears as white point (salt noise). When the black image in a window is corrupted by impulse noises, there are black and white pixels. For an 8-bit image, it means the pixel values are 0 or 255. But most of the pixel values are 0. So $\mathbf{Z_{min}}$ = $\mathbf{Z_{med}}$ = 0, $\mathbf{Z_{max}}$ = 255. Even the window size is increased to maximum, $\mathbf{Z_{min}}$ = $\mathbf{Z_{min}}$ =0, $\mathbf{Z_{max}}$ = 255. In this case, the output value m'n med max should be 0. But, according to the algorithm, the output value is equal to $\mathbf{Z_{xy}}$. $\mathbf{Z_{xy}}$ may be any value between 0 and 255. So the salt noise can not be filtered in the black background of corrupted image. For the same reason, the pepper noise can not be removed in the white background.

If the output value is changed from $\mathbf{Z_{xy}}$ to $\mathbf{Z_{med}}$ in the expression (1), the above questions can be solved. For the black image, $\mathbf{Z_{min} = Z_{med} < Z_{max}}$

output is $\mathbf{Z_{min=} Z_{med=}}$ 0. For ,min <med m Zmed mi

the white image, $\mathbf{Z_{max=} Z_{min} < Z_{med}}$,output is $\mathbf{Z_{max} = Z_{med}}$ = 255. For the general image, $\mathbf{Z_{min} < Z_{med} < Z_{max}}$ the algorithm goes to level B.

The algorithm has three main purposes:

(a) To remove 'Salt and Pepper' noise.
(b) To smoothen any non impulsive noise.
(c) To reduce excessive distortions such as too much thinning or thickening of object boundaries.

6. Result:

Two signals were considered for the test. These were subjected to salt and pepper noise of unit amplitude. Fig. shows one form of the evolved architecture of the reconfigurable fabric during operation. Fig. 7 give the deviations of the output with respect to the original value of the signals.
However, pepper noise still can be observed in the black region of the image, such as the connectors at the left-top ofthe image. Figure 7(c) shows the result of the modified adaptive median filter. Pepper noise in the black region of the image is removed efficiently. Improvement over the classical adaptive median

Fig 7.Results of filtering with a 3*X*3 median and conditional median filter. From left to right, first row: noisy image; second row: Adaptive median filter,Modified Adaptive median filter.

CONCLUSION

The Proposed architecture provides the capacity of implementing the reconfigurable framework in a pipelined fashion. The architecture is pipelined which processes one pixel per clock cycle, thus to process an image of size *512* x 512 it requires 0.65 ms when a clock of 100 MHz is used and hence is suitable for image processing. The classical adaptive median filter has some deficiencies: the filtered images remain the positive impulse noise in the black background and the negative impulse noise in the white background. To solve the above questions, a modified scheme is proposed. Experiment results shows the proposed scheme can improve the filtering performance significantly.

References

[1] Zdenek Vasicek, Lukas Sekanina, Novel Hardware Implementation of Adaptive Median Filters 978-1-4244-2277-7/08/ © 2008 IEEE
[2] Olli Vainio, Yrjö Neuvo, Steven E. Butner, A Signal Processor for Median-Based Algorithms, IEEE Transactions on Acoustics, Speech, Processing VOL 37. NO. 9, September 1989.
[3] V.V. Bapeswara Rao and K. Sankara Rao, A New Algorithm for Real-Time Median Filtering, IEEE Transactions on Acoustics, Speech, Processing VOL ASSP-34. NO. 6, December 1986.
[4] M. O. Ahmad and D. Sundararajan, Parallel Implementation of a Median Filtering Algorithm, Int. Symp. on Signals and Systems, 1988.
[5] Dobrowiecki Tadeusz, Medián Szűrők, Mérés és Automatika, 37. É vf., 1989. 3.szám
[6] Xilinx Foundation Series Quick Start Guide, 1991-1997. Xilinx. Inc.
[7] Jim Torresen, "An Evolvable Hardware Tutorial", FPL 2004, 821-830
[8] L. Sekanina, "Evolvable Hardware Tutorial", in GECCO 2007, New York
[9] Bernard Widrow and Samuel D. Steavns, "Adaptive Signal Processing", Pearson Edition, 2000.
[10] Redmill, D. W., Bull, D. R., and Dagless, E., "Genetic synthesis of reduced complexity filters and filter banks using primitive operator directed graphs". IEE Proc. Circuits Devices Syst, vol.147, pp. 303-310, 2000.
[11] Bull, D. R. and Horrocks, D. H., "Primitive operator digital filters", IEE Proc. Circuits, Devices and Systems, pp. 401-412, 1991.
[12] L. Sekanina and P. Mikusek, "Analysis of Reconfigurable Logic Blocks for Evolvable Digital Architectures", EvoWorkshops 2008, LNCS 4974, pp. 144–153, 2008. Evolution Hardware (EH'05)