Journal of VLSI Signal Processing 44, 63–78, 2006
© 2006 Springer Science + Business Media, LLC. Manufactured in The Netherlands.
DOI: 10.1007/s11265-006-5920-3

An Algorithm for Adaptive Mean Filtering and Its Hardware Implementation

IOANNIS GASTERATOS
Laboratory of Electronics, Section of Electronics and Information Systems Technology, Department of Electrical and Computer Engineering, Democritus University of Thrace, GR-67100 Xanthi, Greece

ANTONIOS GASTERATOS
Laboratory of Robotics and Automation, Section of Production Systems, Department of Production and Management Engineering, Democritus University of Thrace, GR-67100 Xanthi, Greece

IOANNIS ANDREADIS
Laboratory of Electronics, Section of Electronics and Information Systems Technology, Department of Electrical and Computer Engineering, Democritus University of Thrace, GR-67100 Xanthi, Greece

Published online: 27 May 2006

Abstract. Noise due to the sensor and the electronics of a camera is an undesirable issue in any machine vision application. Such noise tends to corrupt images and to obstruct any further analysis. An algorithm to detect and cancel such noise, using statistical methods, is presented in this paper. The proposed algorithm is an adaptive mean filter, which filters out image regions that are found to be noise corrupted. The efficiency of the proposed filter was examined both qualitatively and quantitatively, by software simulation in several noisy conditions. The main advantage of the filter in hand is that it is appropriate for hardware implementation and can be easily incorporated to smart cameras. The hardware implementation of the filter is also presented in this paper. This implementation aims at time critical applications such as machine vision, inspection and visual surveillance.

Keywords: Image filtering, Gaussian noise, Real time systems

1. Introduction

In many imaging applications one of the major problems often arising is the noise, i.e. random variations in image intensity values. This can be classified into two main categories [1, 2]: (a) Uniform and (b) non uniform. In the first of the two categories the distribution of the noise is a Gaussian one and it can be suppressed by means of a linear filter, whilst in the second one the distribution is not normal and a non-linear or hybrid filter (such as the a-trimmed mean filter) should be applied. The uniform noise is a very good model to many kinds of sensor noise, such as noise due to camera electronics [3].

A filter can be viewed as an operator that transforms an image into another one more adequate for interpretation. Several linear filters exist, with the most popular one being the mean filter. This is computed as the convolution of the image with the mask:

$$\frac{1}{N \times N}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{1}$$

in case of an $N \times N$ image window.

This operation corresponds to low-pass filtering implemented directly in the spatial domain. As obvious, it eliminates the high-frequency elements, which usually corresponds to noise, but to edges as well. In other words the mean filter and other linear filters tend to blur the image, reducing, therefore, its quality by withdrawing valuable information. The same drawback stands for contemporary filtering techniques (e.g. [4, 5]), that elaborate more sophisticated methods than a simple window averaging. However, the aim of an efficient noise filtering would be to suppress the noise and its effects on the original image, while being as little alterative for the image in hand as possible. To this end, the necessity of an adaptive filter that is applied only onto the corrupted image regions is straightforward. Such a filter requires a decision technique and a criterion whether a certain pixel is noisy or not. In the literature several such filters have been reported for both uniform [6–9] and non uniform [10–12] noise suppression. The decision criteria may vary, including statistical methods [6, 7], fuzzy inference systems [8, 9] or threshold operations [10–12].

Only few of the above mentioned techniques are hardware oriented [6, 11, 12]. Due to the fact that hardware implementation requires simple structures, the more sophisticated the filter and the decision rule, the more difficult to be implemented in hardware. In this paper a new adaptive mean filter and its hardware implementation are proposed. The decision criterion is the well-known statistical criterion of 'Chi-Square Goodness-of-Fit Test' [13]. This detects the existence of Gaussian noise in the image and applies a linear mean filter. The proposed filter is, therefore, more efficient than any conventional filter as it cancels the noise in areas that exists, using the mean filter, avoiding the blurring of the image in areas that filtering is not necessary. The application of the mean filter on the noisy image areas was chosen because of its simplicity and its speed when implemented in hardware. This is to say that the algorithm is hardware oriented and, thus, its FPGA implementation is also presented. The proposed implementation is intended to be used cascaded to an image sensor, so that it suppresses the noise due its electronics. Therefore, issues such as hardware complexity and processing time are crucial. Moreover, apart the simplicity and speed that the linear filters exhibit, they are the most appropriate ones for Gaussian noise type cancellation, such as the one due to camera electronics, than non-linear filters, which are appropriate for impulse and other non uniform noise [14]. The rest of the paper is organized as follows: The proposed algorithm is presented in section 2. Some basic concepts are defined first and the restrictions introduced by the digital implementation are analyzed. Experiments carried out to exhibit the efficiency of the algorithm are shown in section 3. The digital hardware implementation of the algorithm in FPGA is given in section 4 and, finally, concluding remarks are made in the last section.

2. Algorithm Description

2.1. Basic Definitions

- Gaussian distribution:

A Gaussian, or normal, distribution is a symmetrical frequency distribution, which is described by a parametric function relating the mean value m and standard deviation of σ the samples.

$$p_{m,\sigma}(x) = \frac{e^{-0.5\left(\frac{(x-m)}{\sigma}\right)}}{\sigma \cdot \sqrt{2 \cdot \pi}} \tag{2}$$

The normal distribution is a valuable statistical tool because it describes a majority of events that occur in the real world in a random way [13].

- Chi Square test:

The chi-square test is a statistical test, which is used to examine whether a sample of data follows a specific distribution. It provides a measure of the distance between the observed distribution and the theoretical one. The formula for the computation of the chi-square test is:

$$x^2 = \sum_{i=1}^{n} \frac{(O(i) - E(i))^2}{E(i)} \tag{3}$$

where n is the length of the distribution, $O(i)$ is the observed frequency, i.e. the frequency of the sample data to be examined and $E(i)$ is the expected frequency.

A test statistic follows a chi-square distribution with $(k - c)$ degrees of freedom, where the number of non-empty cells is k and the number of estimated parameters, including location, scale and shape parameters is $(c - 1)$. Therefore, the hypothesis that the data derive from a population with the specified distribution is rejected when $x^2 > x^2_{a,k-c-1}$, with $x^2_{a,k-c-1}$ being the chi-square percent point function having $k - c$ degrees of freedom and a significance level of a.

In the proposed algorithm, the chi-square test was used to determine whether a grayscale histogram of an image matches the shape of the Gaussian distribution. The workflow of the algorithm was followed with slight alterations which facilitate its implementation in a digital circuit, as it is described in the following paragraphs.

2.2. *Restrictions*

Several restrictions are imposed by the requirement that the algorithm is easily realizable in a digital circuit.

- **Complicated operations:** The digital circuit itself introduces restrictions related to the complexity of several operations. Complicated arithmetic operations must be avoided, since they are both time and hardware consuming. Such operations are the square root, the logarithm, the exponential and even the division. On the other hand the addition and the multiplication are considered to be rather simple.
- **Speed:** This is another important issue. The main objective is the achievement of real time processing. This limitation determines that it should be a one-pass algorithm, i.e. no pre-processing can be done, or no global characteristics can be calculated for the whole image.
- **Memory:** The whole image can not be stored in a local memory. This would require a large amount of memory and would introduce a limitation to the size of the images.

2.3. *Algorithm Description*

The proposed algorithm determines the existence of noise in a local area of the digital image, by examining the shape of the distribution of the grayscale values in that specific area. It uses the chi-square test to extract a measure of the similarity of the above distribution with the Gaussian distribution. Since the random noise follows the Gaussian distribution, if the similarity lies bellow a certain threshold the existence of noise is traced.

A square window is defined with dimensions $N \times N$ around each image pixel x_i. The mean value m and the standard deviation σ of the grayscale values in that area are calculated:

$$m = \frac{\sum_{i=1}^{N\times N} x_i}{N\times N} \tag{4}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N\times N} (x_i - m)^2}{N\times N}} \tag{5}$$

If the examined area in the image is distorted significantly by random noise, then the distribution of the grayscale values in that area will have the shape of a Gaussian distribution, which is described by the above parameters. The chi-square test is the tool that is used to measure the similarity between the data distributions and the Gaussian one:

$$x^2 = \sum_{i=1}^{N\times N} \frac{(Dist[i] - E_{m,\sigma}(i))^2}{E_{m,\sigma}(i)} \tag{6}$$

where, $Dist[i]$ is the distribution of the grayscale values in the window, $E_{m,\sigma}(i)$ is the function of the Gaussian distribution with mean and standard deviation m and σ, respectively. The Gaussian distribution $E_{m,\sigma}(i)$ is the product of the Gaussian probability $p_{m,\sigma}(x)$ and the population of the data in the window, which is $N \times N$. The result of the chi-square test is a monotonically increasing function of the probability with which the sample of data follows a certain distribution, which in our case is the Gaussian. This probability can be calculated directly from the x^2. However, there is no reason to perform such a calculation in digital circuitry, since this would require additional time and a more complicated circuit. Instead, the x^2 value itself, is compared to a threshold value, below which it is assumed that there is a noteworthy resemblance between the window and the Gaussian distribution or, in other words, there is a significant amount of noise in the window. The threshold value determines the strictness of the decision and it is a subjective criterion. If the judgment is positive then a mean filter is applied to the window, otherwise it remains unaffected. This procedure is repeated for each pixel until the whole image is processed. The output of the algorithm is a new image array where only the noise corrupted pixels are filtered out, whilst the noise-free pixels are not altered.

3. Software Implementation

The proposed algorithm was implemented in software and several tests were performed with a large set of

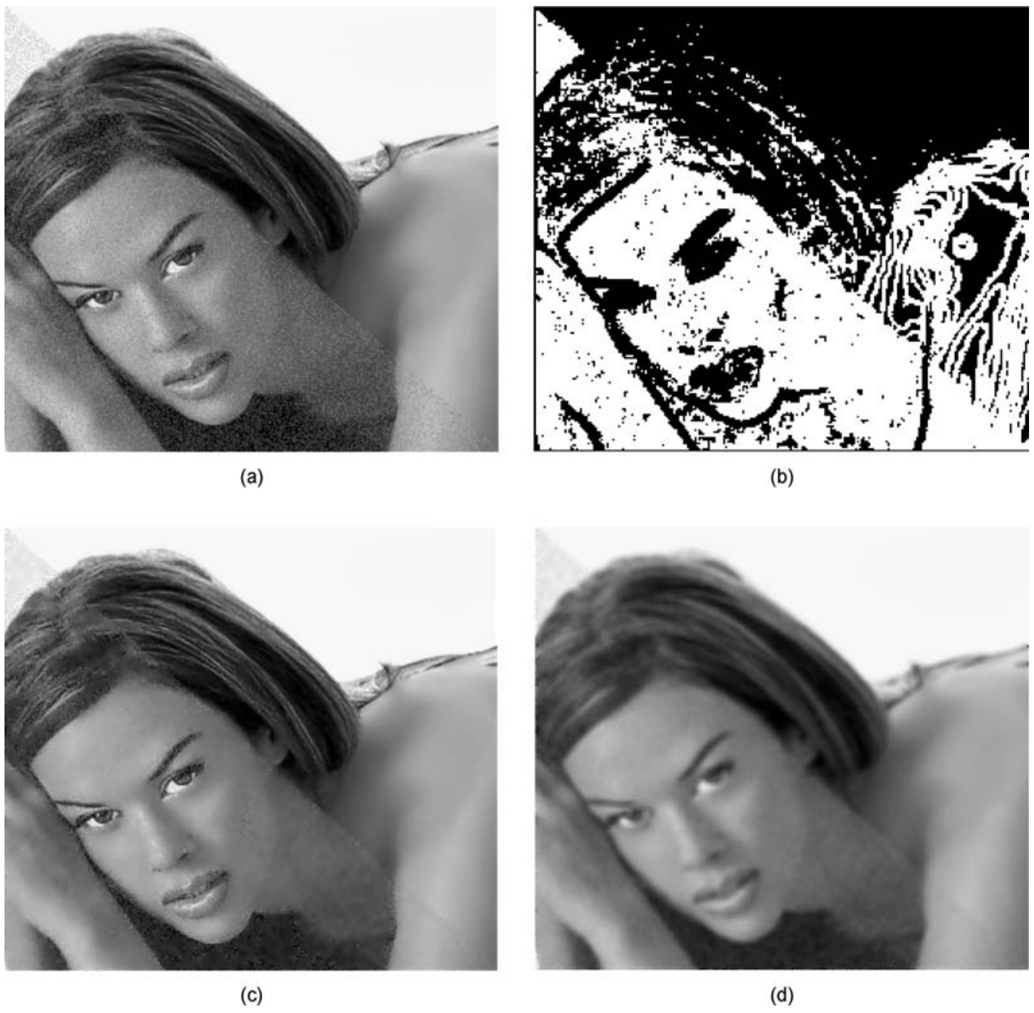


Figure 1. Demonstration of the simulation process: (a) Noise is applied to the half image area, (b) the area where noise has been detected (in white); the noisy area has been successfully detected, (c) the results of the proposed algorithm; a mean filter is applied only to the pixels that are detected to be noise corrupted, and (d) the results of a global mean filter applied to the whole image area; here the edges have been distorted.

images. A Windows application was developed for this purpose, which provides the capability to test the performance of the algorithm on a wide range of conditions and to examine the effect of the various parameters, such as the window size, the noise characteristics and the criterion threshold-value.

The first step of the processing is to apply random noise on the original image. The type of noise may vary in strength and in density and, thus, all the possible conditions can be reproduced. The next step is to calculate the chi-square test value for each pixel, as described in section 2. Finally, these values are compared with a fixed threshold value. The pixels which have a value lower than the threshold are considered to be corrupted by random noise, and an $N \times N$ mean filter is applied to them to suppress the noise. The size of the window can be parameterized.

Figure 1 demonstrates the simulation process. The first conclusion of this procedure is that the algorithm recognizes successfully the areas where Gaussian noise was applied. Quantitative measures were performed on the filtered image, which include the extraction of the MSE and the PSNR between the original and the final image. A set of images was used as input, and uniform noise of 10% strength and 60% density was applied to their area. Then, both the proposed algorithm and a global mean filter were applied. The window size that was used in both methods was 5×5-pixel and the

Table 1. PSNR values. Comparison between the proposed algorithm and the standard mean filter for different types of images.

File	Description	Mean filter	Proposed adaptive mean filter
Model.bmp	Smooth image, clean of noise.	29.16	35.60
Photograph.bmp	Photograph from a digital camera. Corrupted by CCD noise	29.78	33.78
Autumn.bmp	High resolution noiseless image with many details.	30.71	39.24
TextImage.bmp	Scaned text document. Corrupted by scanner noise.	22.70	38.40

threshold value equal to 7. The comparison between the two methods, for several well-known images, is shown in Table 1. As it can be observed, the proposed algorithm outperforms the classical global mean filter.

Another desirable feature of the proposed algorithm is that it is sensitive to the areas where edges dominate or there are high intensity fluctuations. It is preferred not to apply a mean filter in such areas, even if they contain random noise, since this would result to blurriness of the edges. However, the intensity distribution in these areas would not match the Gaussian distribution, even if they are noisy. The reason is that in such areas the local histogram possesses at least two distinct lobes. These are formed by the intensity variation due to the edge that leads to a non-uniform intensity distribution. As a result the value of the chi-square test will be high and this area will be excluded from filtering. The proposed algorithm will give positive results, if the noise intensity is significant in comparison to the existing intensity distribution in a certain area. This simulates the way the human vision perceives a scene. An example of the performance of the algorithm in an image with strong edges is shown in Figure 2.

The threshold value is an important parameter in the behavior of the proposed algorithm, which defines the severity of the algorithm. The lower the threshold values the less the random noise removal from all the areas it occurs. Nevertheless, if the threshold value is chosen to be too high, areas which have not been affected by noise might be filtered. Therefore, there is no optimal value for this parameter which deals all kinds of random noise. However, some limitations can be defined. As it can be observed in Figure 3, extremely high threshold values should be avoided because they do not perform well, according to the MSE and PSNR metrics. On the other hand a high PSNR value on the diagrams should not interpreted as a better result for the human observer.

One final remark is that the threshold value grows with the window size. The proposed threshold values are 18 for a 7 × 7-pixel window, 7 for a 5 × 5-pixel window and 3 for a 3 × 3-pixel window. These values have been resulted through experiments and produce fine results for the majority of noise conditions. However, these values may vary depending on the desirable results and on the image characteristics.

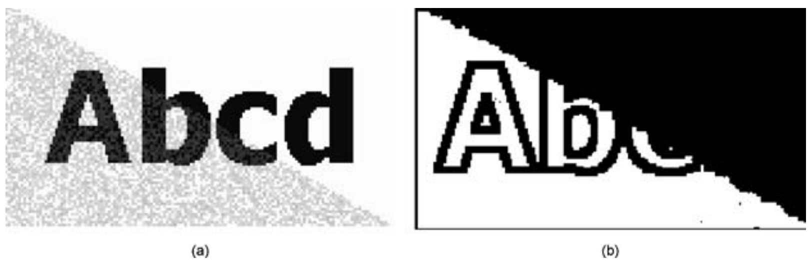


Figure 2. The results of the algorithm when applied to an image with strong edges: (a) text image with noise applied to the half of the area and (b) the area where the chi-square test is positive; the edges have been excluded and no filtering is applied to them.

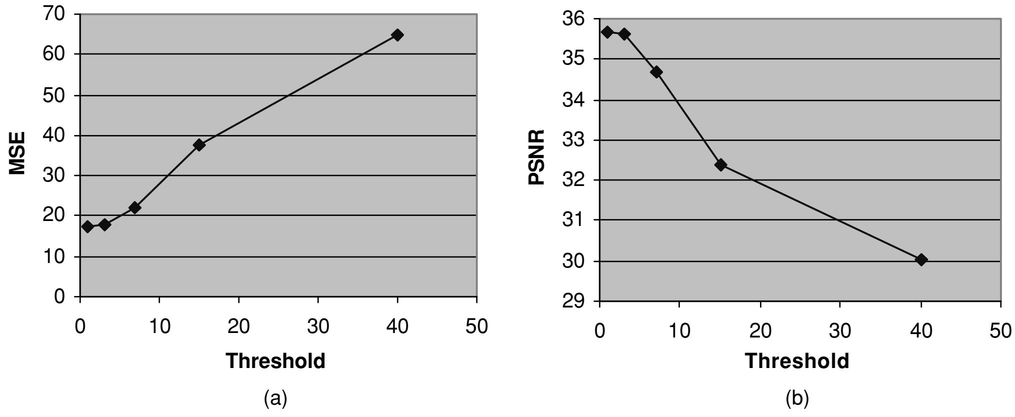


Figure 3. The effect of the threshold value on the quality of the final image: (a) MSE values in relation to the threshold value and (b) PSNR values in relation to the threshold value.

Table 2. NMSE values. Comparison between the proposed proposed algorithm the mean filter, the Wiener the FLS and the clustering fiter.

Gaussian noise with standard deviation s:	NMSE ($\times 10^{-2}$)				
	Mean filter	Wiener filter	FLS filter	Clustering filter	Proposed algorithm
5	1.20	1.53	1.02	1.09	1.07
10	1.90	2.63	1.46	1.60	1.65
15	2.54	3.58	1.98	2.30	2.38

The proposed algorithm is also compared with two contemporary algorithms which have been developed for Gaussian noise reduction. The first algorithm is a filter to suppress Gaussian noise, based on clustering gray levels (Clustering filter) [7]. It runs a clustering algorithm called CLOSE (Clustering by Local Separation) to divide the window's histogram into separable areas and then it applies a mean filter to the cluster that contains the central pixel. The second algorithm is structured around a selective feedback fuzzy neural network (SFNN). A fuzzy logic system (FLS) is defined which is aimed on suppressing Gaussian noise [9]. Table 2 shows a comparison of the performance of the above algorithms in the terms of Normalized Mean Square Error (NMSE). The NMSE value of the results of two classical linear filters for noise canceling, such as the mean filter and Wiener filter, are also presented. Figures 4 and 5 depict the outputs of the above filters applied to a sample images corrupted with Gaussian noise. As it can be observed, the proposed algorithm outperforms the classical linear filters, but though it is comparable to the FLS and the clustering ones it is not as effective as these two. However, these filters comprise highly effective sophisticated algorithms that are very difficult to be implemented in hardware, and to achieve high computation speeds. Thus these algorithms are not hardware oriented as the proposed one and, therefore, they are not appropriate for the target applications.

4. Hardware Implementation

The digital circuit is composed by five logical blocks. Each block executes a specific logical task. These blocks are fully pipelined, and they operate in parallel. The five high-level blocks are:

1. The 'Serpentine Memory' block
2. The 'Distribution' block
3. The 'Deviation' block
4. The 'Chi-Square Test' block
5. The 'Decision' block

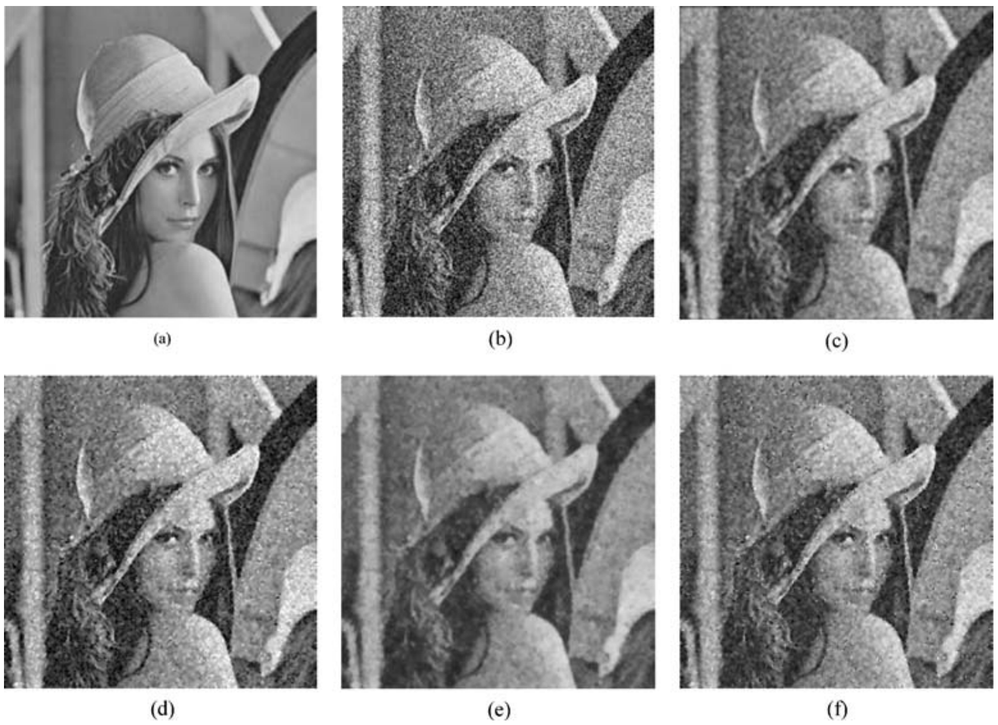


Figure 4. A comparison of the proposed filter with other filters on the image of Lena: (a) Original image; (b) image corrupted with Gaussian noise with standard deviation $\sigma = 7.6$; (c) mean filter; (d) clustering filter; (e) FLS filter and (f) the proposed filter.

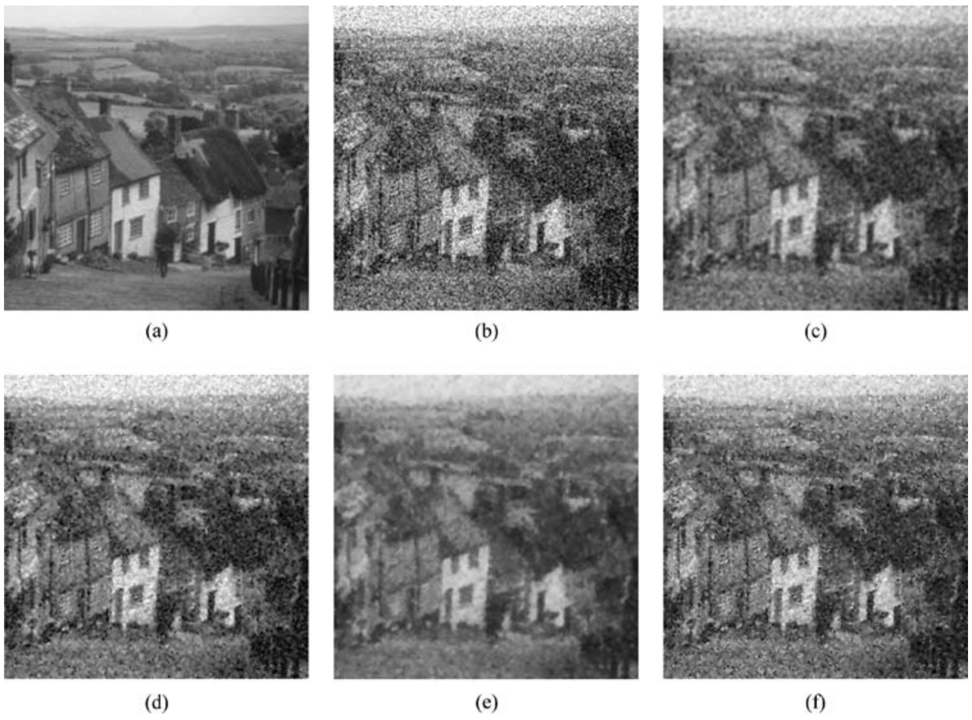


Figure 5. A comparison of the proposed filter with other filters on the image Goldhill: (a) Original image; (b) image corrupted with Gaussian noise with standard deviation $\sigma = 7.6$; (c) mean filter; (d) clustering filter; (e) FLS filter and (f) proposed filter.

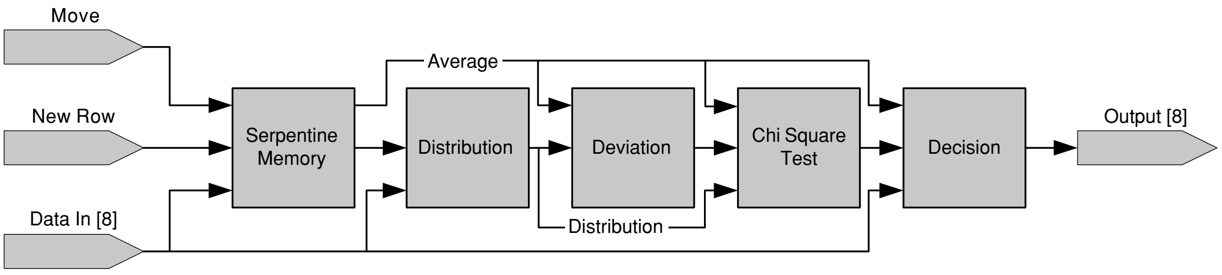


Figure 6. The block diagram of the proposed system.

Figure 6 presents the block diagram of the digital system.

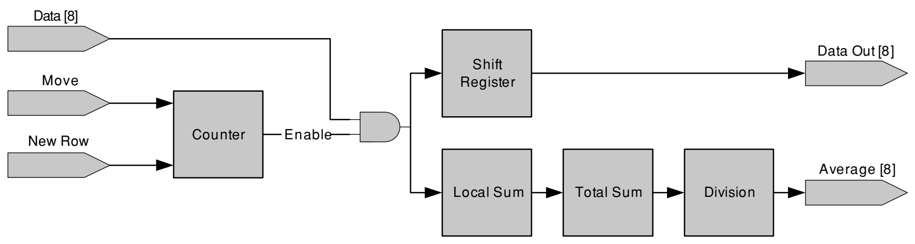
4.1. The 'Serpentine Memory' Block

The 'Serpentine Memory' is the input block of the system. Its block diagram is depicted in Figure 7-a. It temporarily stores the grayscale values of the pixels and it provides the circuit interface. It also performs a real time pre-processing on the data, which involves the calculation of their summary and their average value. The circuit does not store the whole image in a local memory, but it only stores the pixels which are contained in the active window. These pixels are transferred into the 'Serpentine Memory' block through the input channel 'Data In $\{8\}$', one in each clock cycle. As the window moves over the image, the 'Serpentine Memory' must be fed with the new pixels. Two cases are distinguished: (a) the active window moves from one pixel to its right neighbor in the same row; (b) the active window moves to the first pixel of the next row. In the first case the two neighbor pixels have overlapping windows, which differ by only N pixels, i.e. the width of the working window. Only these pixels must be fed into the circuit. In the second case however, the whole window must be refreshed, which means that $N \times N$ pixels must be fed into the circuit. These actions are controlled by signals 'Move' and 'New Row', which enable input bus 'Data $\{8\}$' for N and $N \times N$ clock cycles, respectively. The pixels are stored in a shift register of $N \times N$ length. The data flow out of the shift register through the output signal 'Data Out $\{8\}$'. The average value of the data is calculated in real time. A block –labeled 'Local Sum'—calculates the sum of a column of N pixels while block 'Total Sum' holds in an array of registers the sum of the last N columns and calculates the total sum of their values. This technique has the advantage that as the window moves over the image, only the summary of the new pixels is calculated while the previous pixels have already been summed and stored into a local memory. This technique requires fewer adders and needs less time to complete. Finally, the sum is divided by the window pixel population ($N \times N$) to compute the average of the grayscale values in the window. The use of the average value is twofold: first it is an important statistic parameter of the data that participates into the oncoming calculations; and second it is the result of a mean filter.

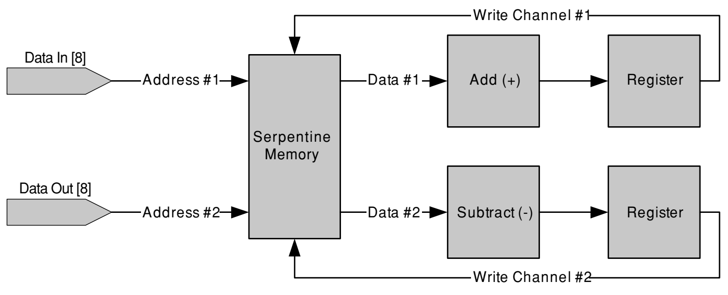
4.2. 'Distribution' Block

The 'Distribution' block is the second cascade block of the circuit. It forms and stores the distribution of the grayscale values of the pixels that belong to the active window. While the grayscale values may vary from 0 to 255, the distribution receive integer values into the interval [0, 32], which means that eight continuous grayscale values are grouped into a new one. To achieve this, only the five MSBs of the grayscale vector are utilized. The reason is that for small window sizes, the distribution would not have a solid shape and the chi-square test would perform poorly.

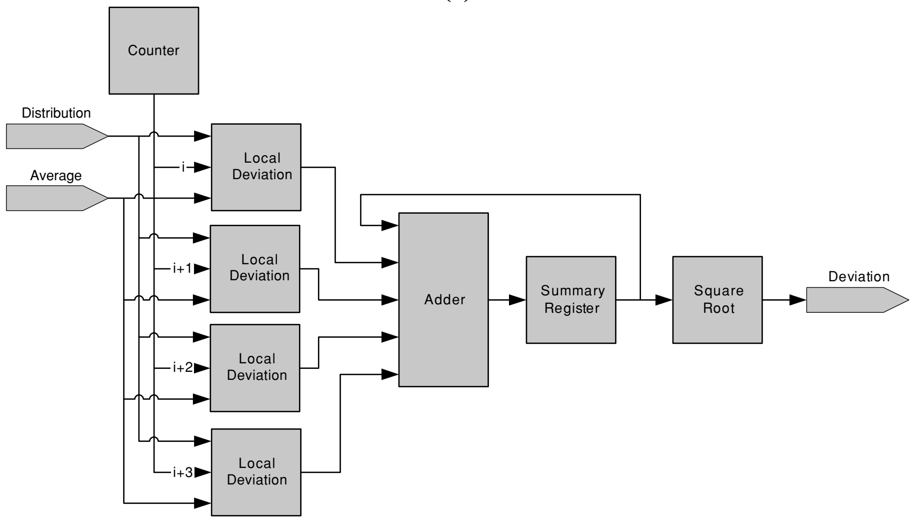
The main element of the distribution block is a dual port memory, which holds the distribution array. The two main inputs of the block are the grayscale value of the pixel that flows into the window ('Data In $\{8\}$') and the grayscale value of the pixel that shifts out of it ('Data Out $\{8\}$'). These drive the two address buses of the memory. The first one selects the appropriate distribution cell, which is increased by one and then is written back into the memory. The second one lessens its corresponding distribution cell by one, as it corresponds to a pixel that does not belong to the active window anymore. The reading of the memory is asynchronous while the writing is synchronous.



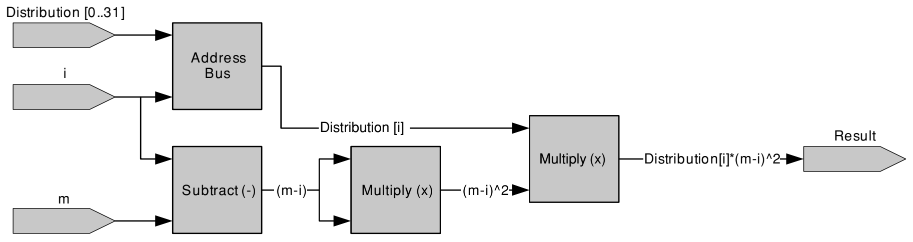
(a)



(b)

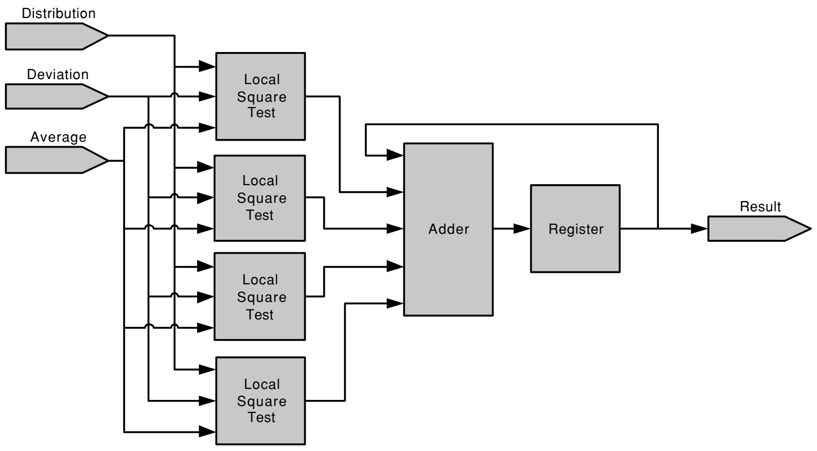


(c)

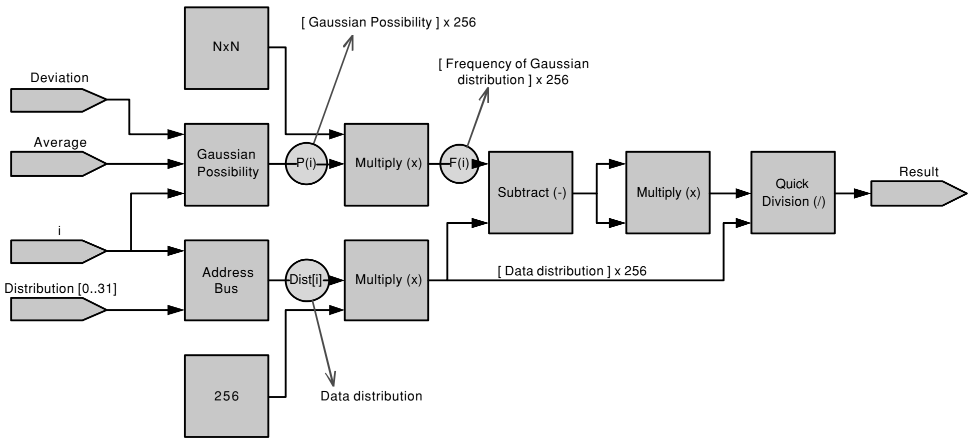


(d)

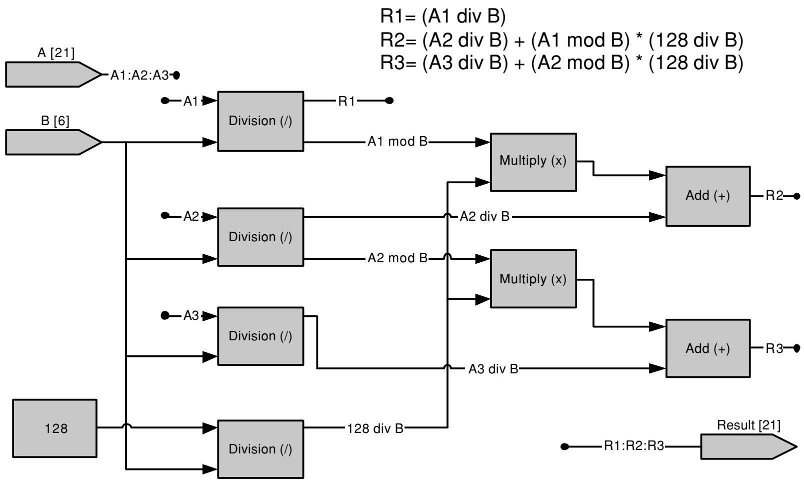
Figure 7. Structural blocks of the circuit: (a) The 'Serpentine Memory' block, (b) the 'Distribution' block, (c) the 'Deviation' block, (d) the 'Local Deviation' block (*continued*)



(e)

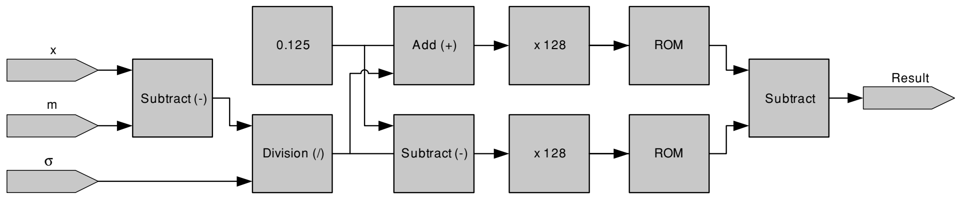


(f)

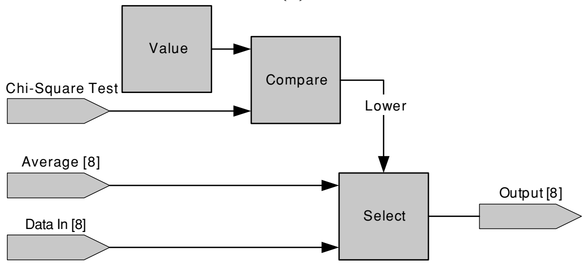


(g)

Figure 7. (*Continued*) (e) the ‘Chi- Square Test’ block, (f) the ‘Local Square Test’ block, (g) the ‘Quick Division’ block (*continued*)



(h)



(i)

Figure 7. (*Continued*) (h) the 'Gaussian Probability' block and (i) the 'Decision' block.

Special treat is taken when the values of 'Data In {8}' and 'Data Out {8}' are equal. In this case the whole block is secluded and no operation occurs. Also, the two input buses are not valid all the time. Special signals which are supplied by the 'Serpentine Memory' block enable each of them for as long as they are valid. These signals are auxiliary and are not presented in the block diagram shown in Figure 7-b.

4.3. *'Deviation' Block*

The deviation block calculates the standard deviation of the contents of the window. The calculation of the standard deviation requires 32 iterations, which are independent from the dimensions of the data window. This is calculated using the contents of the distribution memory and not these of the 'Serpentine Memory' block:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N\times N}((i-m)^2\cdot Dist[i])}{N\times N}} \tag{7}$$

where m is the average value, $Dist[i]$ is the distribution array, $N\times N$ is the population of the data, and i obtains values between 0 and 31.

Four concurrent sub-blocks named 'Local Deviation' complete the calculation as depicted in Figure 7-c. These blocks calculate the quantity $(i-m)^2\cdot Dist[i]$ for the different values of the parameter i. They are simple to implement and their inputs are the distribution array, the average value and the value of the parameter i. An adder with feedback calculates the sum. In each clock cycle, four values are calculated by the blocks 'Local Deviation' (Figure 7-d), and the results are added to the contents of the 'Summary Register'. The initial value of that register is set to zero. A modulo eight counter generates all the 32 possible values of parameter i. The total number of clock cycles that are needed to complete that calculation is 32/4 = 8 clock cycles.

Finally the division and the square root of the expression are calculated. This is a rather complicated operation, which is executed by a specialized block named 'Square Root' block. This calculates the result of the division by $N\times N$ and the square root of a value $\sqrt{x/(N\times N)}$, using a successive approximation process. The required clock cycles to complete this operation are equal to the number of bits of the result. In each clock cycle one bit is calculated, starting from the MSB and ending to the LSB. The pseudo-code of the algorithm that calculates the root of a function $f(x)$ is:

```
Set $x = 0000000$
Begin Loop
    Set the MSB $\rightarrow 1$
    If $f^{-1}(x) > 0$ then set the MSB $\rightarrow 0$
    Move to the next bit
Repeat until the LSB
```

The strong feature of the above algorithm is that it does not have to calculate the value of the complicated function $f(x) = \sqrt{x/(N \times N)}$ but of its inverse $f^{-1}(x) = x^2(N \times N)$, which is much simpler to implement, and faster to execute. The maximum length of the result is 7 bits, which is also the delay in clock cycles that this block introduces.

4.4. 'Chi-Square Test' Block

The 'Chi-Square Test' block, the block diagram of which is shown in Figure 7-e, implements the chi-square test as described in eq. (6). This is the most complex block of the total system, as it performs complicated mathematical tasks. The value of the Gaussian probability (1) must be calculated in each clock cycle, which involves the calculation of a square root and an exponential. Again, there should be executed 32 iterations over the contents of the distribution array. The quantity that should be calculated in each iteration is:

$$G(x) = \frac{(Dist[i] - E_{m,\sigma}(i))^2}{E_{m,\sigma}(i)} \tag{8}$$

This calculation is executed for the different values of the parameter i by the specialized sub-block named 'Local Square Test'. Four instances of this block operate in parallel to reduce the delay in clock cycles, as demonstrated in Figure 7-e. An adder with feedback sums the results of all the 32 calculations. The 'Local Square Test' block is one of the most complex blocks of the system. It calculates the quantity $G(x)$ during one clock cycle, as illustrated in Figure 7-f. While the distribution values of the data in the memory are directly available, the value of the theoretical Gaussian distribution $E_{m,\sigma}(i)$ has yet to be computed. A look up table (LUT) has been implemented in a separate sub-block, labeled 'Gaussian Probability', to perform this task. This block calculates the value of the Gaussian probability $P_{m,\sigma}(i)$ with parameters m and σ, and returns its value normalized by a factor of 256. This is the minimum factor for a reliable precision of the operations. All the other quantities that participate to the calculations are also multiplied by 256 and that holds for the output of the block too. Due to the restrictions in calculations time, the final division was chosen to be approximant. A special block ('Quick Division') was designed, which calculates the result of the division with an accuracy of about 97% in almost the half time than the normal division.

A distribution was created and loaded to 'Distribution' signal, to perform the timing simulation of 'Local Square Test' block, as shown in Figure 8-a. This is a distribution of 49 values which corresponds to a 7 × 7-pixel window and it exhibits the characteristics of a Gaussian distribution. Figure 8-b shows the results of the timing simulation for a value of signal i equal to 21. The 'Local Square Test' block is a sequential circuit. The time that is required until the output is stabilized is 95 nsecs, which is the maximum delay introduced into the pipeline process and, therefore, it determines the maximum frequency of operation.

4.5. 'Quick Division' Block

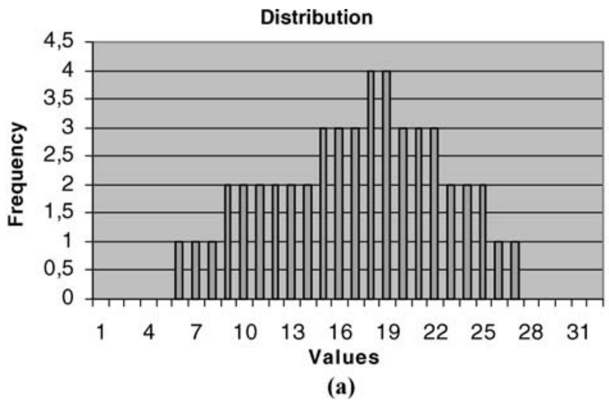
The quick division block, shown in Figure 7-g, performs the division between dividend 'A {21}' and divisor 'B {6}'. Due to the big length of the dividend the normal division would be extremely time-consuming. This circuit splits the dividend into three parts of 7 bits each (A1, A2 and A3), it performs some parallel processing and then recomposes them to form the value of the quotient of the division. It is in fact the implementation of the algorithm of the division in the 128dic system of numeration:

$$\begin{aligned} R1 &= (\,A1 \; div \; B\,) \\ R2 &= (\,A2 \; div \; B\,) + (\,A1 \bmod B\,) \cdot (128/B) \\ R3 &= (\,A3 \; div \; B\,) + (\,A2 \bmod B\,) \cdot (128/B) \end{aligned} \tag{9}$$

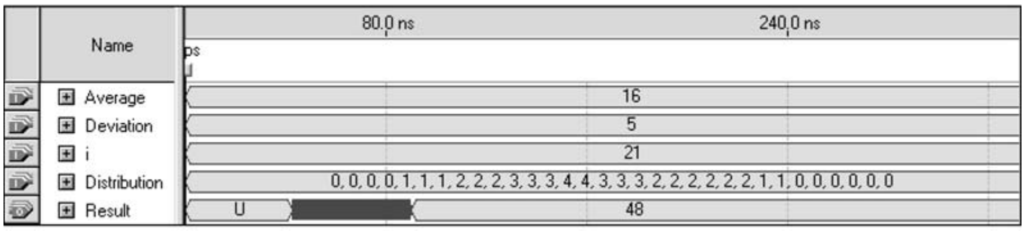
The reason that it is not absolutely precise is that the decimal division (128/B) was replaced with the integer division (128 div B).

4.6. 'Gaussian Probability' Block

The 'Gaussian Probability' block, returns the value of Gaussian probability (eq. (2)) with parameters m and σ at value x. The utilization of a LUT was preferred because the direct calculation would involve complicated mathematical operations. However the function of Gaussian probability can not be stored directly in a LUT because it is a function of three variables (x, m, and σ). This would require a three-dimensional LUT which is not easy to implement and would require a large amount of memory and, consequently, a great silicon area on the physical mean. An auxiliary function was stored instead, from which the desirable result can be acquired with simple calculations [13]. The algorithm flow is described below:



(a)



```
Average       16
Deviation     5
i             21
Distribution  0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 3, 3, 3, 2, 2, 2, 2, 2, 2, 1, 1, 0, 0, 0, 0, 0, 0
Result        U                48
```

(b)

Figure 8. 'Local Square Test' block timing simulation: (a) The distribution that was used as input and (b) the timing simulation results.

We define the function:

$$\Phi(x) = \int_0^x e^{-\frac{\omega^2}{2}} d\omega \tag{10}$$

The probability that a fact x occurs in the interval $[a, b]$ is:

$$\begin{aligned} p(a < x \leq b) &= \int_a^b p(x)dx \\ &= \Phi\left(\frac{b-m}{\sigma}\right) - \Phi\left(\frac{a-m}{\sigma}\right) \end{aligned} \tag{11}$$

with $\Phi(-s) = 1 - \Phi(s)$.
For an integration length equal to d we get the following approximation:

$$p(x_o) \approx \frac{p(x_o - d/2 < x \leq x_o + d/2)}{d} \tag{12}$$

and from eq. (11):

$$p(x_o) = \frac{\Phi\left(\frac{x_o - m}{\sigma} + d/2\right) - \Phi\left(\frac{x_o - m}{\sigma} - d/2\right)}{d} \tag{13}$$

The integration length d was selected to be equal to 0.25, which achieves a satisfactory resolution and keeps the quantization error low. The function $\Phi(x)$ was stored in a LUT, and the implementation of eq. (13) is performed by a specialized block shown in Figure 7-h. Two instances of the LUT were stored in separate ROMs, so that the two references to the value of function $\Phi(x)$ can be accessed in the same clock pulse. The values of $\Phi(x)$ were calculated with a step of 1/128, multiplied by a factor of 1024 and stored into the memory. The requested addresses were also scaled by 128 before applied to the memory. It must also be noted that the final division by d (=0.25) is omitted. This results in a total scaling factor of 256 at the output of the block. These transformations are taken into account in the later stages. The selected ROM with a 9-bit address bus and 10-bit data bus possesses a size of 0.625 Kbytes. The results of this block are enough accurate. The maximum deviation from the theoretical value of the Gaussian probability is 0.004, whilst the typical error is no greater than 0.001. The comparison of the theoretical value of the Gaussian curve and the value that is calculated by this block are shown in Figure 9.

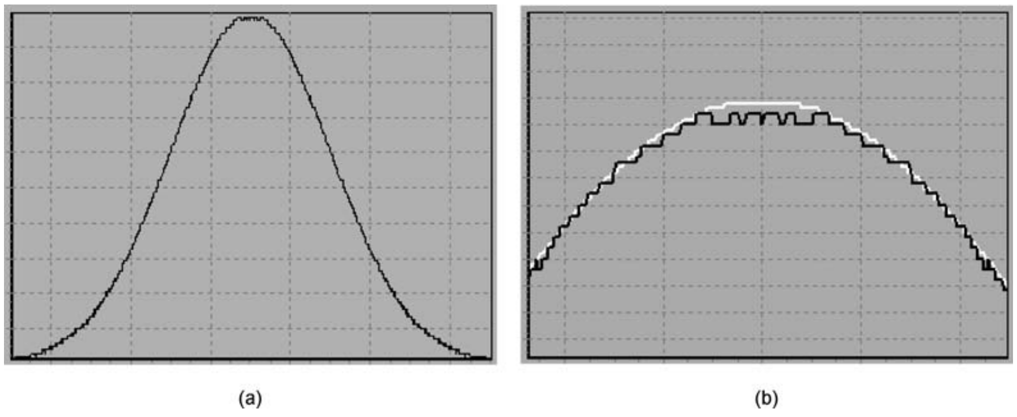


Figure 9. Approximation of the Gaussian probability by means of a LUT: (a) The Gaussian Curve as it is approximated by a LUT, (b) a detail of case (a). The theoretical Gaussian curve is represented in light color while the calculated through a LUT values are represented in dark color.

4.7. 'Decision' Block

This block, presented in Figure 7-i, decides whether the current pixel has been affected by noise and it assigns a value to the output accordingly. It compares the result of the 'Chi-Square Test' block with a predefined threshold value. If it is lower, then it assigns to the output the average value of the current window, calculated in the 'Serpentine Memory' block), otherwise the output is the value of the current pixel unmodified.

4.8. Circuit Characteristics

The digital system described above, was designed, simulated and implemented with the 'Quartus II' package provided by Altera. The target FPGA was EP1C12F256C8 device from the 'Cyclone' family. The size and timing characteristics of the circuit being presented below are software and hardware dependent, i.e. they might vary substantially from one device to another. The selected FPGA is considered to be slow in comparison with the other FPGAs of the same family. Tables 3 and 4 present the size of the circuit in logic cells and in memory bits, respectively. One logic cell is the unit of size in the Altera's FPGAs series and it usually implements one or two logic gates. The window size is an important parameter for the design of the digital circuit that has an immediate effect on the total resources (both Logic Cells and Memory bits) that will be required. Some blocks, such as the 'Serpentine Memory', the 'Distribution' and the 'Deviation' grow proportional with the window area, while blocks 'Chi-Square Test' and 'Decision' are fixed size. The overall relationship between window size and the total resources that are required is shown in Figure 10.

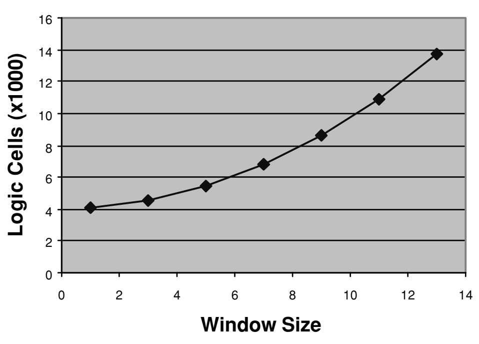
The delays that each block introduces to the pipeline are: the 'Serpentine Memory' block: (Window width) + 1 clock cycles, the 'Distribution' block: 1 clock cycle, the 'Deviation' block: 11 clock cycles, the 'Chi Square Test' block, 12 clock cycles and the 'Decision' block: 1 clock cycle. As it is obvious, the slower block determines the overall speed of the system. 'Chi-Square Test' block is the slower block. The maximum frequency of operation is 10.53 MHz. One processing cycle for one pixel consists of 12 clock cycles (for window widths smaller than 12 pixels). This leads to a

Table 3. Resources utilization of the whole digital circuitry in comparison with the total available resources in the device.

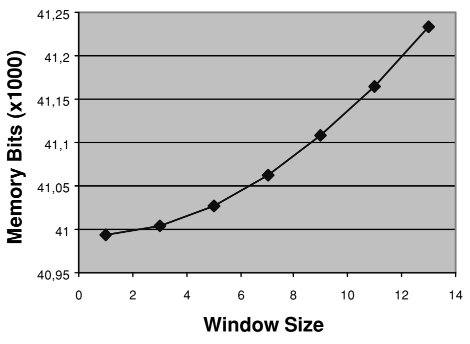
	Logic cells	Memory bits
Resources utilization	6.840	41.030
Available in the device	12.060	239.616
Percent (%)	54%	17%

Table 4. Resources utilization of each block.

Block	Logic cells		Memory bits	
Serpentine Memory	683	10.0%	70	0.17%
Distribution	587	8.6%	0	0.00%
Deviation	1.539	22.6%	0	0.00%
Chi Square Test	3.968	58.3%	40.960	99.75%
Decision	24	0.4%	32	0.07%



(a)



(b)

Figure 10. The scaling of the digital circuit in relation to the window size: (a) Total logic cells and (b) memory bits that would be required.

total time of 1140 nsecs per pixel or a processing rate of 880.000 pixels per second. For example a 512 × 512-pixel image would need 300 msecs to be processed, which is equal to a frame rate of 3.3 fps. The maximum image size that can be processed in a real time rate of 25 fps is 35.200 pixels which correspond to an image with dimensions 190 × 190 pixels. Higher processing speeds can be achieved with more blocks connected in parallel where each block will scan a different line of the image. The upper limit of processing blocks is the number of lines of the image, and the processing speed is directly proportional to the number of blocks.

5. Conclusions

A new algorithm for canceling random noise suppression in grayscale images and its implementation in a digital system was presented in this paper. The key feature of this algorithm is that it detects the areas on the image which are corrupted by random noise, whilst it does not process the noiseless areas. Also, the algorithm does not distort areas containing sensitive data, such as edges. The mathematical tool that was used to detect the existence of noise was the chi-square goodness of fit test. The efficiency of the proposed algorithm, both quantitative and qualitative, was experimentally demonstrated. The proposed algorithm was compared both to two classical linear filters and to two contemporary ones (FLS and clustering filter) aimed to Gaussian noise suppression. The results show that it outperforms the classical filters, whilst it is comparable to the more sophisticated contemporary filters, being more modest though by means of the NMSE measure. However the proposed algorithm was designed to be hardware oriented, whilst the other two algorithms pay more attention to the effectiveness of the results than to implementation efficiency. The proposed algorithm is an optimal trade-off between these two aspects. A digital circuit implementing the proposed algorithm also proposed. The circuitry is fully pipelined and is optimized for speed. It supplies a simple interface to operate, and it can be easily embedded in larger systems. Its main applications are found in the field of digital sensors and real time image processing.

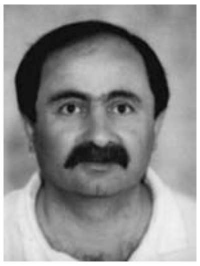
References

1. I. Pitas and A.N. Venetsanopoulos, *Nonlinear Digital Filters: Principles and applications*, Boston: Kluwer Academic Publishers, 1990.
2. R.C. Gonzales and R.E. Woods, *Digital Image Processing*, Addison-Wesley, Reading, MA, 2002.
3. R. Jain, R. Kasturi and B.G. Schunck, *Machine Vision*, Singapure: Mc Graw-Hill, 1995.
4. O. Demirkaya, "Smoothing Impulsive Noise Using Nonlinear Diffusion Filtering," in *Lecture Notes in Computer Science*, M. Šonka et al. (Eds.), Springer-Verlag Berlin Heidelberg, vol. LNCS 3117, 2004, pp. 111–122.
5. L. Bar1, N. Sochen and N. Kiryati1, "Image Deblurring in the Presence of Salt-and-Pepper Noise" in Lecture Notes in Computer Science, R. Kimmel, N. Sochen and J. Weickert (Eds.), Springer-Verlag Berlin Heidelberg, vol. LNCS 3459, 2005, pp. 107–118.
6. A.M. Borneo, L. Salinari and D. Sirtori, "An Innovative Adaptive Noise Reduction Filter for Moving Pictures Based on Modified Duncan Range Test," *ST Journal of System Research*, vol. 1, 2001, pp. 14–22.
7. Z. Atae-Allah and J. Martinez Aroza, "A Filter to Remove Gaussian Noise by Clustering the Gray Scale," *J. Mathematical Imaging and Vision,* vol. 17, 2002, pp. 15–25.
8. M. Muneyasu, K. Asou, Y. Wada and T. Hinamoto, "Edge-Preserving Fuzzy Filters Based on Differences between Pixels".

Electronics and Communications in Japan, vol. 81, 1998, pp. 47–55.
9. S.T. Wang, F.L. Chung, Y.Y. Li, D.W. Hu and X.S. Wu, “A New Gaussian Noise Filter Based on Interval Type-2 Fuzzy Logic Systems,” Soft Computing, vol. 9, 2005, pp. 398–406
10. E. Abreau, M. Lightstone, S.K. Mitra, and K. Arakawa, “A New Efficient Approach for the Removal of Impulse Noise from Highly Corrupted Images,” *IEEE Transactions on Image Processing*, vol. 5, 1996, pp. 1012–1025.
11. G. Louverdis , I. Andreadis and N. Papamarkos, “An Intelligent Hardware Structure for Impulse Noise Suppression,” *Int. Conf. on Signal Processing & Applications*, September 2003, Rome , Italy, pp. 483–488.
12. G. Louverdis, I. Andreadis and A. Gasteratos, “A New Content Based Median Filter,” *12th European Signal Processing Conference (EUSIPCO 2004)*, September 2004 Vienna, Austria, pp. 1337–1340.
13. A.H.S. Ang and W.H. Tang, *Probability Concepts in Engineering Planning and Design*, vol. 1, Wiley, 1975.
14. C. Kotropoulos, *Design and Implementation of Non-Linear Filters for Digital Image Processing*, Ph.D. Thesis, Aristotelian University of Thessaloniki, Thessaloniki 1993.

Ioannis Gasteratos holds a Diploma in Electrical Engineering from the Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece, 2004. His research interests include digital VLSI design, computer architectures and artificial intelligence. He is a member of the IEEE, and a member of the Technical Chamber of Greece (TEE).

Antonios Gasteratos is a Lecturer of Robotics in the Department of Production and Management, Democritus University of Thrace, Greece. He holds a PhD from the Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece, 1999. During 2001–2003 he was a visiting Assistant Professor in the Department of Electrical and Computer Engineering, Democritus Univesrsity of Thrace. He serves as a reviewer to numerous of Scientific Journals and International Conferences. His research interests are mainly in computer and robot vision and sensory data fusion. He is a member of the IEEE, the IAPR, the EURASIP, the Hellenic Society of Artificial Intelligence (SETN) and the Technical Chamber of Greece (TEE).



Ioannis Andreadis received the Diploma Degree from the Department of Electrical & Computer Engineering, DUTH, Greece, in 1983 and the MSc and PhD Degrees from the University of Manchester Institute of Science & Technology, UK, in 1985 and 1989, respectively. His research interests are mainly in Intelligent Systems, Machine Vision and VLSI based computing architectures. He joined the Department of Electrical & Computer Engineering, DUTH in 1993. He is a member of the Editorial Board of the Pattern Recognition Journal, TEE and IEEE.