J Sign Process Syst
DOI 10.1007/s11265-016-1187-5

A Novel Image Impulse Noise Removal Algorithm Optimized for Hardware Accelerators

Vivek Singh Bhadouria[1] · Alexandru Tanase[2] · Moritz Schmid[3] · Frank Hannig[2] · Jürgen Teich[2] · Dibyendu Ghoshal[1]

Received: 15 December 2015 / Revised: 9 September 2016 / Accepted: 22 September 2016

Abstract Images are often corrupted with noise during the image acquisition and transmission stage. Here, we propose a novel approach for the reduction of random-valued impulse noise in images and its hardware implementation on various state-of-the-art FPGAs. The presented algorithm consists of two stages in which the first stage detects whether pixels have been corrupted by impulse noise and the second stage performs a filtering operation on the detected noisy pixels. The human visual system is sensitive to the presence of edges in any image therefore the filtering stage consists of an edge preserving median filter which performs the filtering operation while preserving the underlying fine image features. Experimentally, it has been found that the proposed scheme yields a better Peak Signal-to-Noise Ratio (PSNR) compared to other existing median-based impulse noise filtering schemes. The algorithm is implemented using the high-level synthesis tool PARO as a highly parallel and deeply pipelined hardware design that simultaneously exploits loop level as well as instruction level parallelism with a very short latency of only few milliseconds for 16 bit images of size 512 × 512 pixels.

Keywords FPGA · Image denoising · Noise detection · Parallel architecture · Random-valued impulse noise

1 Introduction

Noise is often introduced in digital images during the image acquisition and (or) transmission stages, thereby degrading the image quality and causing loss of image details. As a result, it is difficult to extract any relevant information from such noise corrupted images. This effect is clearly observed in case of medium- and high-level image processing operations such as, edge detection, image segmentation, object detection, recognition, etc. are severely affected by the presence of noise in the images. Therefore, noise reduction is of vital importance in the image processing applications.

Impulse noise is mainly caused due to malfunctioning of the Charge-Coupled Device (CCD) imaging sensors, transmission errors, and faulty locations in the memory storage device [14]. In this context we can differentiate between two different classes, namely (1) saturated value impulse noise, also known as salt-and-pepper noise, and (2) random-valued impulse noise. Pixels corrupted by salt-and-pepper noise can take either 0 or 255 gray-value (for any 8-bit grayscale image). However, random-valued impulse noise corrupted pixels exhibit random distribution and thus can

✉ Vivek Singh Bhadouria
vivek@hotmail.de

Alexandru Tanase
tanase@cs.fau.de

Moritz Schmid
moritz.schmid@siemens.com

Frank Hannig
hannig@cs.fau.de

Jürgen Teich
teich@cs.fau.de

Dibyendu Ghoshal
tukumw@gmail.com

[1] Department of Electronics and Communication Engineering, National Institute of Technology, Agartala, India

[2] Hardware/Software Co-Design, Department of Computer Science, Friedrich-Alexander University Erlangen-Nürnberg (FAU), Erlangen, Germany

[3] Siemens Healthcare GmbH, Erlangen, Germany

acquire any intensity value in the allowed grayscale range (0–255 for any 8-bit image) [5, 29]. It is difficult to define a robust criteria for the detection of impulse noise corrupted pixels because of their random distribution. Various effective approaches have been proposed for the reduction of salt-and-pepper noise [6, 11, 29]. However, designing an effective random-valued impulse noise reduction method remains a challenge because noise corrupted pixels can take any value in the dynamic range of the image. Classical approaches compare the central pixel value with its neighboring pixels to determine the central pixel's nature. However, such approaches yield high false classification rates because they inevitably detect edge pixels as noisy pixels. In the present work, we will concentrate our discussion to random-valued impulse noise only.

The contributions of this work can be summarized as follows.

- We propose a novel algorithm for image impulse noise removal with a comparatively better Peak Signal-to-Noise Ratio (PSNR), compared to other existing noise reduction algorithms.
- We propose a novel noisy pixel detection algorithm, which relies on iteratively reducing the detection criteria. Detection criteria here refers to the approach used for classifying a pixel as noisy pixel or noise-free pixel.
- We present a highly parallel and deeply pipelined hardware implementation of the proposed algorithm.
- We map our noise reduction algorithm using a High-Level Synthesis (HLS) framework that yields VHDL code, optimized for hardware accelerators.

The organization of the article is as follows. Background and related work is discussed in Section 2, the proposed noisy pixel detection algorithm and filtering operation are given in Section 3. Section 4 describes the hardware implementation in detail. Simulation results are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2 Background

Several non-linear filtering schemes have been proposed for the restoration of images, corrupted with the random-valued variant of impulse noise. Among them, one of the most widely used filters is the Standard Median Filter (SMF) [18]. Mathematical simplicity, ease of use and high computational run-time efficiency of the median filters has led to its undisputed hegemony for the image restoration processes. SMF utilizes the rank-ordered information of the pixels in the sliding window and replaces the central pixel with the median of the gray-value of the pixels contained in the corresponding local window. SMF performs effectively at low noise density. However, at higher noise densities, SMF blurs the fine image details resulting in a blurred restored image. Adaptive Median Filtering (AMF) was proposed to overcome the drawbacks of SMF in [19]. Although AMF performs satisfactorily at low noise densities, the filter shows inefficient performance at high noise densities, for which the window dimension has to be increased for the denoising operation. Furthermore, increasing window dimension will lead to blurring of the image details along with a high run-time complexity [11]. Other variants of SMF viz. Weighted Median Filter (WMF), Multi-State Median (MSM) filter, stack filter, etc., were proposed to achieve better noise removal performance. However, these algorithms set the same filtering process parameters for noise-free as well as for noisy-pixels. As a result, the filtering operation also alters the noise-free pixel's intensity, leading to restored images that are blurred.

Several switching median filters have been proposed to overcome the effect caused by excessive filtering. The switching median filtering process consist of two stages, namely: (1) noisy pixel detection stage, and (2) filtering stage. Stage one involves an impulse detector whose function is to flag pixels either as noisy or noise-free, with a follow up filtering operation on the detected noise pixels in stage two. Such schemes perform the filtering operation only on the detected noisy pixels of the corrupted image, thereby reducing the distortion caused due to filtering operation on the noise-free pixels. Apart from the median filters, other types of filters were also proposed for the purpose of impulse noise reduction. Luo proposed an Alpha Trimmed Mean Based Method (ATMBM) in which the alpha trimmed mean was employed for impulse noise detection and the noisy pixel was replaced by a linear combination of the original value and the median of the neighboring pixels [21]. Aizenberg and Butakoff proposed a Differential Rank Impulse Detector (DRID) in which the impulse detector unit compares the pixel value within a narrow range of the rank window by both, rank and absolute value [2]. Zhao and Wang proposed a Rank Order Relative Differences (RORD) statistic to identify whether a pixel is noise contaminated or not [36]. Dong and Wu introduced a Directional Weighted Median (DWM) filter in which noisy pixel detection is performed by calculating the difference with four aligned neighbor pixels in a 5×5 window, followed by a weighted median filtering based denoising [9]. Schulte, et al., presented a two stage impulse noise reduction method based on type-1 fuzzy logic [27]. Petrovic and Cmojevic proposed an impulse noise reconstruction filter based on a switching scheme with two cascaded detectors and two corresponding estimators [24]. Genetic programming was used as a supervised learning scheme for building the two cascaded detectors for impulse noise detection based on the criteria of median and MAD (Median of Absolute Deviation from the median). Lien et al. proposed a decision tree based

approach for the impulse noise detection, followed by an edge preserving image filter [20]. This algorithm assumes many predefined thresholds which limit its application in various imaging conditions.

The existing impulse noise detectors use a local statistical measure to find the presence of noisy pixels, e.g., the simplest approach is to compare the pixel's gray-value with the median of the value of the neighborhood pixels. However, existing state-of-art denoising algorithms use more intricate criteria to determine the pixel's nature [9, 10, 32]. Therefore, denoising algorithms can be broadly classified into two classes, depending upon their search criteria for impulse noise corrupted pixels and the denoising approach, namely (1) low complexity algorithms, and (2) high complexity algorithms [20]. Lower complexity denoising algorithms generally employ a fixed window size of 3×3 pixels. Such algorithms are very efficient due to the small window size but also yield visually as well as quantitatively poor results. High run-time complexity denoising algorithms use a complex approach involving adaptive search approaches (in which the search window is expanded locally in order to find noise- free pixels for the reconstruction of the central pixel), and involve a large number of mathematical operations to yield visually noise-clean images. However, such high complexity approaches require long computational time, which is not suitable for the real-time requirements of consumer electronics. Some researchers also reported efficient median filtering implementation that can be of vital importance for consumer electronics [22, 35, 37].

Many of today's real-time consumer electronic products incorporate a denoising unit for a better customer experience. Therefore, there is an increasing demand for efficient, real-time denoising methods. In the present work, we propose a novel two stage algorithm consisting of basic mathematical operations, e.g., addition and division. By using simple mathematical operations in impulse noise corrupted pixel detection and noise reduction phases, the computational complexity can be reduced, resulting in a low latency hardware implementation. In the present algorithm, the first stage consists of a noisy pixel detection mechanism based on the difference between the central pixel value and the mean of its neighbor pixels. After the detection of noisy pixels, the second stage consists of a filtering operation in which the detected noisy central pixel is replaced by the reconstructed value obtained from a gradient based edge preserving median filtering operation in a fixed window of dimension 3×3 pixels.

3 Noisy Pixel Detection and Noise Reduction

The proposed noise reduction scheme consist of two stages: (a) an impulse noise detector based on a sliding mean vector approach followed by (b) a gradient based edge preserving median filter, shown in Fig. 1. A more detailed algorithm flow diagram has been shown in Fig. 2. The purpose of the noise detector based on a sliding mean vector is to determine whether the central pixel, $X(i, j)$ is noisy or noise-free in nature (see Section 3.2). If the central pixel is found to be noise-free then it is left intact. In contrast, if the pixel is found to be a noisy pixel, the gradient based edge preserving median filter reconstructs the value of the noisy pixel by approximation based on the method discussed in Section 3.4. In the present work, the following random-valued impulse noise model has been considered to develop a spatial filter for the restoration of noisy images [28]. The description of the noise model is presented as follows in Section 3.1.

3.1 Random-valued Impulse Noise Model

Unlike additive noise where noise is added to every pixel of the image, random-valued impulse noise corrupts only a portion of the pixels in the image [25]. Let P be the probability of occurrence of random-valued impulse noise in any clean image, I. Mathematically, the model is given as:

$$X(i, j) = \begin{cases} n(i, j) \text{ withprobability} P \\ I(i, j) \text{ withprobability} 1 - P \end{cases} \quad (1)$$

where, $X(i, j)$ and $n(i, j)$ are the resultant and noisy image, respectively. For any 8-bit grayscale image, random-valued noise corrupted pixels can acquire any grayscale intensity value in the gray-range of $[L_{\min}, L_{\max}]$ where $L_{\min} = 0$ and $L_{\max} = 255$ for 8-bit image.

It is to be noted here that noise model presented in Eq. (1) applies to impulse noise only. In the case of Gaussian noise (which is a type of additive noise), $P = 100$ % i.e. noise is added to every pixels of the image. It is worthwhile to

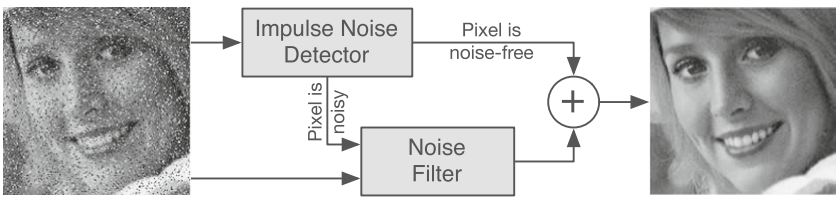


Figure 1 Proposed noise reduction scheme.

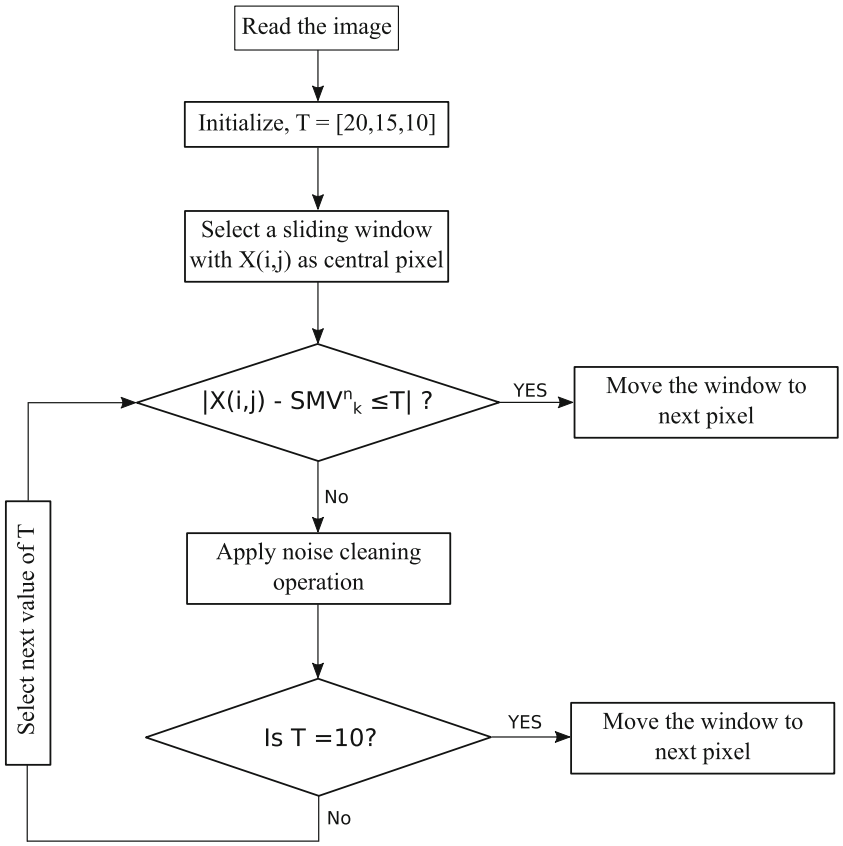


Figure 2 Algorithm of the proposed noise reduction scheme.

observe that at $P = 100\ \%$, requirement of any noisy pixel detection stage becomes unnecessary. Since, every pixel in the additive noise corrupted image is noisy, therefore median calculation for the purpose of filtering will again yield a noisy reconstructed value. Therefore, the present algorithm is not applicable to additive noises. We restrict our further discussion to random-valued impulse noise only.

3.2 Sliding Mean Vector Approach for Noisy Pixel Detection

In this section, we propose a Sliding Mean Vector (SMV, i.e., the vector formed by the aggregation of mean values) approach for the detection of pixels corrupted by impulse noise in any noisy image.

Let $X\,(i,\,j)$ represent the intensity of a pixel at any spatial location $(i,\,j)$ of the input noisy image X of dimension $(M \times N)$ where $i\,(1 \leq i \leq M)$ and $j\,(1 \leq j \leq N)$ are the row and column indices, respectively. Let $W\,(m, n)$ denote the set of pixels contained in a window, centered at an arbitrary spatial location $(i,\,j)$ of the input image X, with a dimension of 3×3.

$$W\,(i,\,j) = X\,(i + m,\,j + n)\,|m, n \in [-1,\,1] \tag{2}$$

In order to compute the SMV, $W\,(i,\,j)$ can be sorted and represented by a vector through rearranging the 2-D matrix as:

$$W_{\text{sorted}-\text{vect}}\,(i,\,j) = [X_1,\,X_2,\,X_3.....X_9] \tag{3}$$

where, $X_1 \leq X_2 \leq X_3 \leq ..... \leq X_9$. The $k - th$ $\text{SMV}^{\text{n}}_{\text{k}}$ for n pixels[1] can be obtained from $W_{sorted-vect}\,(i,\,j)$ as:

$$\begin{aligned}
\text{SMV}^3_{\text{k}} &= \text{mean}\,(X_k,\,X_{k+1},\,X_{k+2})\,,\,(k < 8)\\
\text{SMV}^4_{\text{k}} &= \text{mean}\,(X_k,\,X_{k+1},\,X_{k+2},\,X_{k+3})\,,\,(k < 7)\\
\text{SMV}^5_{\text{k}} &= \text{mean}\,(X_k,\,X_{k+1},\,X_{k+2},\,X_{k+3},\,X_{k+4})\,,\,(k < 6)
\end{aligned} \tag{4}$$

The SMV can be visualized as a similarity indicator with varying degree of neighborhood support, i.e. k. By degree of support, we mean the number of pixels utilized to compute the SMV. It is well known from the literature that pixels exhibit a high degree of correlation with their neighbors except in the case of the presence of an edge [20]. In other words, a noise-free pixel will be correlated to its

[1] for the sake of brevity of the text, we have mentioned SMV for $3 \leq n \leq 5$.

neighbors in case of a smooth region, otherwise it will be less correlated if it encounters an edge. Low correlation is also obtained in case of a noisy pixel. Therefore, our target is to exploit the condition of the occurrence of low correlation in order to identify a noisy pixel, depending upon its neighbors' intensities. To evaluate the nature of the pixel, we have considered the *mean* intensity of the neighbor pixels because *mean* provides a better approximation than the median [34] and, moreover, other measures of central tendency such as *median* have a tendency to preserve the noisy outliers [1], thus it may estimate a noisy SMV_k.

Experimentally, we have studied the effect of variation of the number of elements n on the detection results, for a fixed threshold, $T = 10$, over various noise densities of 10 %, 30 % and 60 %, shown in Table 1. A pixel is said to be noisy if its value diverges from SMV_k by more than this threshold value, for any value of k. The evaluation was performed on the basis of Correct Classification Rate (CCR) and False Classification Rate (FCR) that can be calculated as follows:

$$\mathrm{CCR} = \frac{\mathrm{N_D}}{\mathrm{N_A}} \times 100 \tag{5}$$

$$\mathrm{FCR} = \frac{\mathrm{N_F}}{\mathrm{N_T}} \times 100 \tag{6}$$

where, N_D, N_A, N_F and N_T represent the number of detected pixels, the actual number of noisy-pixels present in the image, the number of falsely detected pixels and the total number of pixels present in the image, respectively. It is to be noted here that similar CCR and FCR trends will be obtained for other values of T. Setting $T < 10$ will show high CCR but along with high FCR. In contrary, $T > 10$ will show low CCR along with low FCR.

From Table 1, it can be easily observed that selecting $n = 5$ yields a high CCR at various noise densities but also results in a high FCR which can lead to a blurred restored image. $n = 3$ produces a low FCR but also yields a low CCR. As a result, some of the noisy pixels may be left unprocessed in the image. Compared to $n = 3$, $n = 4$, as listed in Table 1, yields a relatively high CCR over the considered noise range due to the balanced operation in efficiently detecting the noisy pixels and effectively reducing the false classification rate.

Therefore, experimentally we have found $n = 4$ to be the optimum number of elements for the purpose of noisy pixel detection. A pixel is said to be noise-free iff it satisfies the below mentioned criteria for any value of k.

$$|X(i, j) - \mathrm{SMV}_k^4| \leq \mathrm{T} \tag{7}$$

If the above mentioned criteria is satisfied for any value of k, then it signifies that $X(i, j)$ closely resembles (within the allowed threshold T) the average of its neighbors, hence $X(i, j)$ is a noise-free pixel. Eq. (7) can also be explained as a trivial voting mechanism where the pixel is said to be noise-free, if any of the candidates (i.e., SMV_k, for any value of k) votes in favor of the pixel being free of noise.

3.3 Selection of the Threshold T

Determining a threshold for the detection of noisy pixels is a complex process and predominantly determines the quality of the output image. Effect of threshold(s) on the algorithm performance can be determined by two approaches namely: (1) Compare CCR, FCR for each threshold; (2) Compare restored image quantitatively. The first approach suffices if we focus only on a single threshold based detection algorithms however this approach involves complex calculation of missed pixel counts. In the second approach, one needs to apply some standard filtering scheme in order to compare the restored output. In this approach we have used the second approach to determine the threshold values and for this purpose we have used Standard Median Filter (SMF) of dimension 3×3. Our strategy was to employ filtering operation only to the detected noisy pixels corresponding to the candidate threshold.

Existing algorithms utilize a fixed threshold value for the detection operation however our extensive experimentation on threshold(s) illuminates an interesting result that if we

Table 1 Restoration results (in terms of Correct Classification Rate (CCR, in %) and False Classification Rate (FCR, in %)) with different number of sliding elements at varying noise densities ($T = 10$).

Number of	Attribute	Lena			Goldhill			Elaine			Bridge			Aerial		
elements, n		10 %	30 %	60 %	10 %	30 %	60 %	10 %	30 %	60 %	10 %	30 %	60 %	10 %	30 %	60 %
Five	CCR	81.74	73.11	61.86	80.97	72.46	61.76	81.48	73.16	61.70	77.85	70.27	60.48	78.26	70.32	60.60
	FCR	0.39	0.93	4.16	1.19	1.53	4.56	0.76	1.14	4.21	4.64	4.42	6.30	2.50	3.02	6.26
Four	CCR	79.53	68.95	55.41	78.56	68.08	54.38	78.71	68.62	54.93	73.78	64.76	53.14	74.54	65.55	53.29
	FCR	0.23	0.37	2.04	0.57	0.71	2.4	0.37	0.47	2.17	2.59	2.44	3.63	1.66	1.71	3.30
Three	CCR	75.63	63.11	47.01	74.18	61.86	46.52	75.46	63.03	46.88	69.09	58.12	44.72	69.90	59.41	44.98
	FCR	0.09	0.17	0.97	0.20	0.32	1.15	0.12	0.18	0.96	1.37	1.42	2.2	0.92	1.08	1.79

use an iterative approach, we can achieve better PSNR values. Experimentally, we have found to reduce the threshold value after each iteration; From Table 2, it can be inferred that the multi-threshold iterative approach outperforms the uni-threshold approach. Such increase in PSNR can also be due to the multiple-filtering operation on the noise degraded image.

Selecting a large value of T yields a low CCR along with a low FCR. On the other hand, small values of T will result in high CCR along with high FCR. Furthermore, choosing a small value of T will result in filtering many noise-free pixels, which will blur fine image details. Therefore, for a proper filtering, one has to apply a combination of coarse and fine thresholds.

Choosing a large value of T at initial iterations reduces the FCR thereby resulting in significantly high PSNR values (see column corresponding to $T = 20$ in Table 2). However, the number of correctly classified pixels is less for larger values of T. To clean all the noisy pixels, we have found to iteratively reduce the value of T. Iteratively lowering the value of T reduces the false detection rate, even at low values of T (see column corresponding to $T = [201510]$ in Table 2), leading to even a higher PSNR value. It has been observed that the number of correctly classified pixels corresponding to $T = [201510]$ is less compared to correctly classified pixels for $T = 10$ or $T = 20$ evident from PSNR values. Because pixel's nature has been determined by directly comparing the detected noisy image map with the ground truth map. After three iterations, number of pixels were cleaned therefore fewer noisy pixels were left in the image. Number of missed pixels under column corresponding to $T = [201510]$ is also high even after three iterations.

Table 2 Detection results (in terms of dB) with different thresholds at varying noise density (for $n = 4$).

Image	Noise Density (%)	Threshold(s)			
		10	15	20	[20 15 10]
Elaine	10	34.57	33.39	32.06	37.82
	30	24.78	23.48	22.42	30.96
	60	16.93	16.21	15.61	20.28
Lena	10	34.37	33.23	33.02	37.98
	30	24.68	23.35	22.40	30.65
	60	16.82	16.11	15.53	20.16
Aerial	10	29.31	29.93	28.68	30.05
	30	22.03	21.11	20.38	25.56
	60	14.95	14.35	13.91	17.56
Bridge	10	22.81	28.97	28.81	29.18
	30	22.41	21.61	20.90	25.36
	60	15.66	15.04	14.60	18.04
Goldhill	10	33.09	32.28	31.65	35.04
	30	24.04	22.96	21.92	29.31
	60	16.42	15.73	15.21	19.53

Coarse iterations detect the noisy pixels with high impulsiveness only. But for a visually clean image, noisy pixels with small impulsiveness are also cleaned by selecting small values of T. Fine iterations are started, once the coarse iterations are over, so that the false detection rate can be minimized.

The effect of the value of T on the restored image and the need for an iterative threshold is shown in Table 2. Selecting a small value of T (i.e., $T = 10$) results in a high correct-pixel classification along with a high false-pixel classification due to stringent detection criteria leading to lower PSNR values of the restored image. Smaller values of T are capable of even detecting noisy pixels with a small degree of impulsiveness. On the contrary, the correct-pixel classification is low in case of $T = 20$, due to the relaxed detection criteria. A larger value of T has the advantage of yielding a low false-pixel classification rate. In order to harness the capability of detecting noisy pixels with a small degree of impulsiveness of T=10 and the low false classification attribute of T = 20, we have designed the algorithm in an iterative manner in which T is reduced at each iteration. By reducing T at each iteration, false classification can be avoided along with achieving a high PSNR in the restored image. Therefore, we used $T = [201510]$ for the determination of pixel's nature.

We may conclude that the blurring problem can be overcome by selecting sufficiently large values of T. Here, the detection mechanism proceeds by choosing an initial seed of $T = 20$ and henceforth reducing the value of T at each iteration.

3.4 Gradient-based Edge Preserving Median Filter

In the proposed algorithm, after the detection of noisy pixels, the input image (i.e., X) along with the flag matrix (i.e., F, which contains the location of noisy pixels) is fed to the noise-cleaning filter. To develop the proposed filter, we have resorted to a gradient calculation scheme in order to

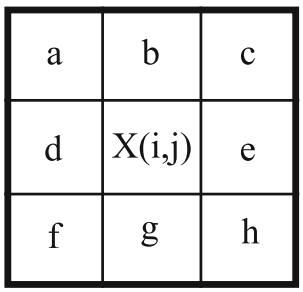


Figure 3 Pixel nomenclature in a 3×3 window.

find the probable presence of edges in the window. The gradient calculation scheme is depicted in Figs. 3 and 4 and mathematical equations are described in Eq. (8). We have considered four directional gradients from $S1$ to $S4$ to calculate the reconstructed value of the noisy central pixel. The significance of using division operation in Eq. (8) is to reduce the effect of noisy pixels (i.e. outliers) on the calculation. Without using division operation (and assuming other operations of Eq. (8) to be intact), any single noisy pixel can surge the respective S calculation, which can further lead to the rejection of that direction [20]. It is important to note that all the pixels were considered to compute the gradient.

Possible patterns of an edge are shown in Fig. 5. Other patterns can also be generated by rotating the illustrated patterns. By observing the patterns shown in Fig. 5, one can easily determine that the gradient calculation scheme depicted in Fig. 4 can efficiently handle all kind of the edges.

$$
\begin{aligned}
S_1 &= \frac{(|f-a|+|g-b|+|h-c|)}{3} \\
S_2 &= \frac{(|a-c|+|d-e|+|f-h|)}{3} \\
S_3 &= \frac{(|g-d|+|h-a|+|e-b|)}{3} \\
S_4 &= \frac{(|d-b|+|f-c|+|g-e|)}{3}
\end{aligned} \tag{8}
$$

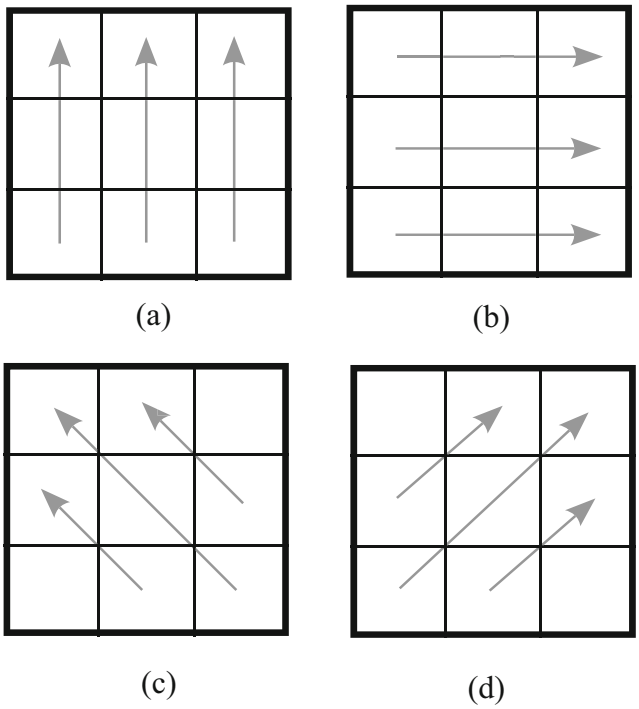


Figure 4 Directional gradient calculation scheme: (**a**) S_1; (**b**) S_2; (**c**) S_3; (**d**) S_4.

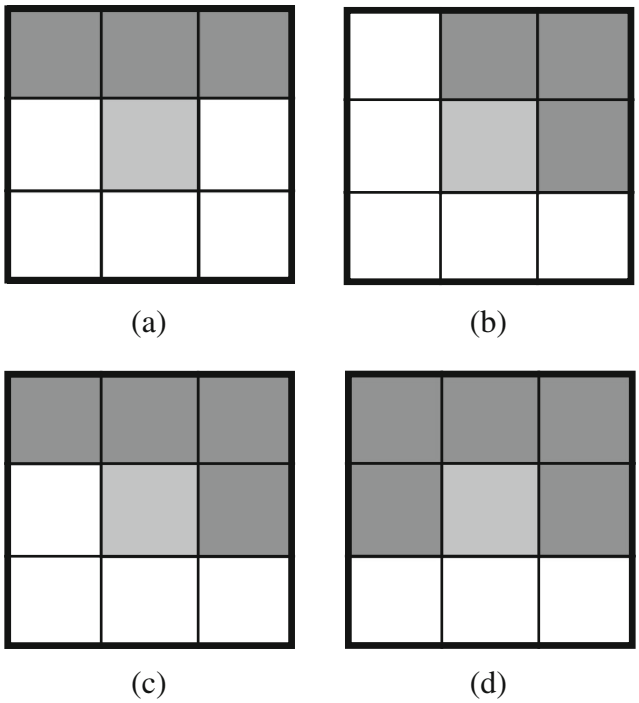


Figure 5 Possible edge locations in a window of dimension 3 × 3. More patterns can be generated by rotating the window.

$$
S_{\min} = \min\{S_1, S_2, S_3, S_4\} \tag{9}
$$

The smallest directional gradient value from Eq. (8) and Eq. (9) represents the set of pixels, having the highest correlation with $X(i, j)$. In order to preserve the edges properly, the filter should replace the noisy pixel with the median of the set of noise-free detected pixels, R and the mean of the pixels, yielding the strongest correlation with $X(i, j)$, $I(i, j)$.

In any window, noise-free gradients can be observed by multiplying the flag matrix with the window (element-wise multiplication) and by observing the non-zero entries in the resulting matrix, as mentioned in Eq. (10).

$$
R = \underbrace{[W] \times [F]}_{\substack{\text{Elementbyelement} \\ \text{multiplication}}} \cap [W] \tag{10}
$$

Mathematically, the reconstructed central pixel's value, $\hat{I}_r(i, j)$ can be calculated using Eq. 12.

$$
\hat{I}(i, j) = \begin{cases}
\frac{(a+b+d+e+g+h)}{6} if\, S_{\min} = S_1 \\
\frac{(a+b+c+f+g+h)}{6} if\, S_{\min} = S_2 \\
\frac{(b+c+d+e+f+g)}{6} if\, S_{\min} = S_3 \\
\frac{(a+c+d+e+f+h)}{6} if\, S_{\min} = S_4
\end{cases} \tag{11}
$$

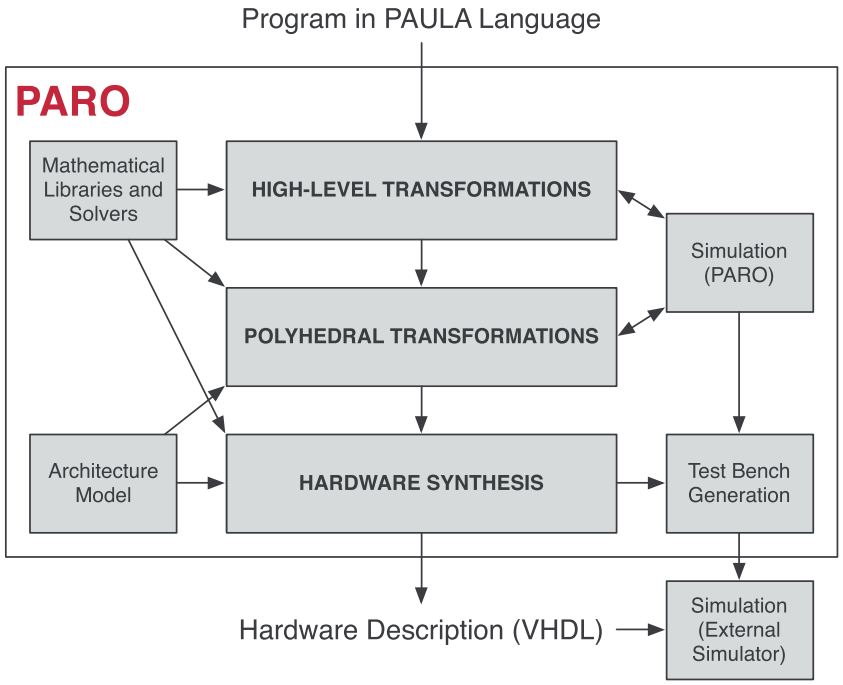


Figure 6 PARO design flow.

$$\hat{I}_r\,(i,\,j) = \text{median}\left[\hat{I}\,(i,\,j)\;R\right] \tag{12}$$

If $\hat{I}\,(i,\,j)$ determined the correct edge then its value will lie at the median of $\left[\hat{I}\,(i,\,j)\;R\right]$. Otherwise, $\hat{I}_r\,(i,\,j)$ will be evaluated from the median of the remaining noise-free pixels.

4 Hardware Implementation

4.1 PARO High-Level Synthesis

Whereas the implementation of simple filter mechanisms in hardware is often trivial, designing a filter module for the proposed denoising algorithm can be rather challenging. Recent years have shown rapid growth in the development of tools for the synthesis of hardware implementations from high-level algorithm descriptions for FPGAs. Many of the leading FPGA and IDE vendors offer their own custom solution, such as Xilinx Vivado HLS [31], the Altera SDK for OpenCL [3], and Catapult C by Calypto Design Systems [7], among others. Moreover, HLS is also researched in academia. For instance, the SPARK [15] synthesis methodology is particularly targeted at control-intensive signal processing applications, however, it can only handle one dimensional arrays. ROCCC 2.0 from Jaquard Computing was originally developed at the University of Riverside at California, USA, [30]. All aforementioned design tools start from a subset of C, C++, or SystemC code, which already determines the execution order of the program. In contrast to the aforementioned approaches, Bluespec [4] is a design system, based on System Verilog, which targets both compute and control intensive applications. Most of the parallelism contained in the original mathematical model of the algorithm is lost during the transformation to sequential code. Although this enables synthesis of sequential hardware, extensive effort must be applied in order to obtain a parallel implementation. The high-level synthesis framework PARO used here [17] allows for the formulation of the algorithm in very close relation to the mathematical description using a functional language, called PAULA [16] and uses the polyhedron model [12] to optimize the program for parallel implementation in the form of a processor array.

The design flow of PARO is depicted in Fig. 6. High-level transformations, which can also often be found in modern software compilers, optimize the program to be represented in the polyhedron model. As a second step, polyhedral transformations are used to restructure the

a	a	b	c	d
a	a	b	c	d
e	e	f	g	h
i	i	j	k	l

Figure 7 The border treatment mirroring for the upper left corner when considering a 3×3 window.

program for parallel implementation as a hardware accelerator. Polyhedral transformations, supported in PARO, include, for example, affine transformations, such as loop reversal, loop interchange, and loop skewing, which are popular instruments for the parallelization of algorithms. Moreover, the polyhedral transformations in PARO perform loop perfectization [33] and loop unrolling to enhance and expose parallelism. Specifically advantageous to signal processing algorithms are transformations to avoid bottlenecks, such as localization, and partitioning of an algorithm. After the program has been suitably restructured, PARO performs a so called space-time mapping to place and schedule the executions. The space-time mapping assigns each iteration point a processor, referred to as allocation and a point in time to execute the iteration. For hardware implementations, it is not sufficient only to determine a global schedule for the iterations. Instead, also a local schedule must be determined to specify the start points for the individual statements of the iteration, which is performed in close relation to the assignment of functional operators to the arithmetic operations of the statements, which is referred to as binding. Hardware synthesis of the restructured program after allocation, scheduling and binding is the last step, which yields a synthesizable hardware description in a language, such as VHDL. For this, PARO must generate descriptions of the processor elements, the interconnection and control structure, as well as the interfaces. During the development process the designer can validate the performed transformations via the built-in simulator, which can also be used to generate a VHDL testbench in order to verify the resulting hardware description in an external simulator.

4.2 Conception of the Filter Module

When dealing with sliding window operations it is important to consider the treatment of pixels at the border of the image, since neighboring pixels, lying outside of the image do not exist. One option is to simply ignore the border pixels which will decrease the output image in size by $\lfloor w/2 \rfloor$ (where w represents the window size) in each dimension. Such a size reduction is often unwanted, since successive sliding window operations would decrease the image size at each turn. Another option is to perform border treatment, for example, by replicating the image border, often referred to as *mirroring*, as shown in Fig. 7.

Another challenge for polyhedron-based high-level synthesis is presented by sorting the elements of Eq. (3).

As reduction operators are a key feature of the PAULA language, the sorting of an array can easily be described by using the SORT reduction operator. The syntax for describing such a reduction operation, is as follows:

```
SORT [iteration_space] (expression);
```

For instance, the sorting of a 3×3 window can be intuitively written in PAULA.

```
par (x >= 0 and x < IMG_X and y >= 0 and y < IMG_Y)
{
  y[x,y] = SORT[k>=-1 and k<=1 and l>=-1 and l<=1](x[x+k,y+l]);
}
```

The first step that PARO must perform for this reduction operator, is to automatically implement a sorting algorithm for the given input space. The two-dimensional sliding-window is linearized and embedded into a four-dimensional data structure, where apart from x and y defining the first two dimensions, the third dimension is used to determine the largest value of the current window and the fourth dimension propagates the minimum between two adjacent values (see Fig. 8 after the sliding-window has been linearized and embedded). PARO automatically inserts the necessary instructions as well as iteration dependent conditions for sorting the given input space.

The next part of the presented denoising filter kernel is the actual computation of the SMV[4] as described in Eq. (4). Traditional programming languages, such as C/C++ or Java, often express summations as a for-loop, predefining a sequential execution order. In PAULA, the summation can be expressed in a very compact and intuitive manner close to the mathematical description by using the SUM operator. This allows for a parallel implementation of the summation, which, if enough resources are available, may reduce the time complexity from linear to logarithmic. The first $SMV_1{}^4$ (see Eq. (4)) for every pixel within the picture over the sorted array may be implemented in PAULA, as follows:

```
par (x >= 0 and x < IMG_X and y >= 0 and y < IMG_Y)
{ ...
  // SMV1 computation
  SMV1[x,y] = SUM[i==8 and j >= 0 and j <= 3 ](sorted_p[x,y,i,j]);
  // i=8 because, the sorted array is available
  // at the end of the 3rd dimension
  ...
}
```

For every noisy pixel, the denoising kernel must compute $\hat{I}$ and $\hat{I}_r$. According to Eq. (11), determining $\hat{I}$ requires the use of an $if \ldots else \ldots$ statement. PAULA supports the implementation of such control statements in the form of runtime dependent conditions, which, for Eq. (11) (where $\hat{I}(i,j)$ = `I[x,y]`) may have the following form:

```
// Syntax: variable = ifrt(cond_exp, then, else)
S_1[x,y] = ifrt(Smin[x,y] == S1[x,y],(p[x,y,3]+p[x,y,7,0]+
                p[x,y,0]+p[x,y,8]+p[x,y,1,0]+p[x,y,5])/6, 0);
...
I[x,y] = ifrt(Smin[x,y] == S4[x,y],(p[x,y,0,0]+p[x,y,2,0]+
               p[x,y,3,0]+p[x,y,5,0]+p[x,y,6,0]+p[x,y,8,0])/6,
               S_3[x,y]);
```

In order to satisfy the single assignment property, both branches must be defined and therefore, the first else branch must receive a zero value. For implementing Eq. (11), a chain of four run-time dependent conditions must be used, where the else branch propagates the earlier computed minimum, so that the last runtime condition will return $\hat{I}(i,j)$. Finally, $\hat{I}_r$ is computed by either finding the median or determining the mean of the uncorrupted pixels in the window, as described in Eq. (12). If the corrupted pixel cannot be reconstructed by the *median*, we need to determine the mean of the elements contained in the current window, excluding zero values. As simply removing zero entries may not be implemented as a regular algorithm and is thus not supported by PARO, we propose a three stage approach: (1) At first, we count the number of zeros inside the window, then (2) the elements inside the window are sorted, such that the zero values are now situated at the end of the array. Following (3) we compute the *mean* values case-wise according to the number of detected zeros.

5 Results and Discussion

5.1 Filter Quality

To study the denoising property and quality of the restored image, simulations of the proposed algorithm were carried out on standard grayscale test images,[2] namely: Lena, Goldhill, Elaine, Bridge and Aerial of dimension 512 × 512. In order to demonstrate the efficacy of the proposed algorithm, we have compared our results to several existing algorithms, i.e., Adaptive Center Weighted Median (ACWM) filter [8], Alpha Trimmed Mean Based Method (ATMBM) [21], Differential Rank Impulse Detector (DRID) [2], Nonlinear Adaptive Video Filtering (NAVF) [13], Low Complexity Noise Removal (LCNR) [23] and Decision Tree Based Denoising Method (DTBDM) [20]. To compare the proposed algorithm with the existing algorithms, quantitative (i.e., PSNR), and qualitative (visual result) evaluation has been used. The mathematical equation for the calculation of PSNR is as follows:

$$PSNR = 10\lg\left(\frac{255^2}{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\left\{I(i,j)-\hat{I}_r(i,j)\right\}^2}\right)\text{dB} \tag{13}$$

where, $I(i,j)$ and $\hat{I}_r(i,j)$ are the noise-free and restored images, respectively. Quantitative results in terms of the PSNR have been listed in Tables 3, 4, 5 and 6 for 5 %, 10 %, 15 %, and 20 % noise density, respectively. Comparing the experimental results of the established algorithms with the algorithm proposed here suggests that the proposed algorithm yields better quantitative results at each noise density, considered in this experiment.

Figure 9a shows a cropped version (of dimension 175×175 pixels) of the standard Elaine image. Figure 9b shows the noise corrupted version of Fig. 9a. Figure 9b was corrupted with 20 % noise density. From Fig. 9c it can be observed that the ACWM approach is unable to remove the impulses, resulting in visible impulses (in the form of dots) in the restored image. ATMBM sufficiently removes the

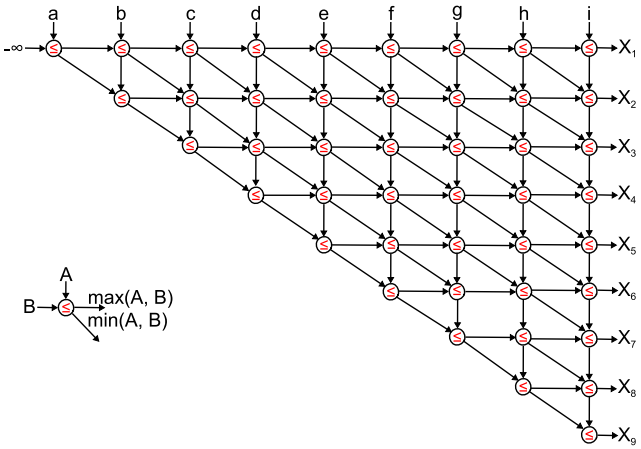


Figure 8 Overview of the systolic sorting algorithm, a linearized 3×3 window is applied as input on the first row, the sorted window can be observed on the last column.

[2] Available as part of the USC-SIPI image database.

Table 3 Quantitative results for images (in terms of PSNR, dB) corrupted with 5 % noise density impulse noise.

Method	Images				
	Lena	Goldhill	Elaine	Bridge	Aerial
ACWM	34.78	31.38	32.66	26.60	27.96
ATMBM	38.83	34.88	39.60	28.43	29.38
DRID	35.56	31.89	33.15	27.16	28.71
NAVF	31.10	28.78	31.70	24.07	24.04
LCNR	39.44	35.13	39.68	28.65	30.24
DTBDM	39.73	35.71	38.66	28.50	30.22
Proposed	40.61	36.86	40.43	30.12	31.15

Table 5 Quantitative results for images (in terms of PSNR, dB) corrupted with 15 % noise density impulse noise.

Method	Images				
	Lena	Goldhill	Elaine	Bridge	Aerial
ACWM	32.66	30.22	31.63	25.67	26.58
ATMBM	34.90	32.31	35.04	26.92	27.55
DRID	33.22	30.54	31.83	26.14	27.09
NAVF	30.53	28.44	31.17	23.77	23.74
LCNR	35.11	32.39	35.06	27.09	28.04
DTBDM	34.74	32.39	34.15	26.78	27.67
Proposed	35.62	33.49	35.14	28.20	28.70

noise from the smooth regions (e.g. skin region) but fails to remove the impulses from the high intensity-variation region which is apparent in the hair region, shown in Fig. 9d. The restored image obtained from DRID method, shown in Fig. 9e contains both positive as well as negative impulses, along with blurring the image details as a result leading to the low PSNR in the restored image. Figure 9e removes all the impulses from the image however, blurs the image detail at the cost of noise reduction. Figure 9g shows the restored image, obtained from the LCNR method. This method performs well in the low variation intensity region, however leaves some traces of noise in the high intensity turbulence areas. A similar observation has also been made with the output obtained from DTBDM, shown in Fig. 9h. DTBDM restore the image details in the smooth areas of the image however, fails to restore the details in the hair region. This can be due to weak impulse detector which in turn yield low correct classification rate. This drop in the performance of decision-tree based based detector can be due to requirement of intricate detection conditions at high noise densities in the image. Figure 9i shows the output obtained from the proposed method. It is apparent from the restored image that the proposed algorithm removes all of the impulses present in the image along with yielding high PSNR, evident from Table 6.

We also investigated the efficiency of the proposed noisy pixel detection method and edge preserving filter towards the reconstruction of noise-free pixels. Table 7 illustrates the quantitative results for the proposed noisy pixel detection scheme with edge-preserving filter and standard median filter at 10 % and 20 % noise density. Please note that only noise-free detected pixels were used in the case of median filter, for the computation of the reconstructed pixel. It can be observed that the edge-preserving filter outperforms the standard median filter. The performance difference is significant at lower noise density (i.e. at 10 %) because there will be more noise-free pixels in neighborhood of the candidate pixel as a result yielding better reconstruction.

5.2 Hardware Implementation and Performance

Following the conception of the proposed algorithm using the PAULA language, PARO synthesizes a VHDL description for the implementation in hardware, for example as a full-custom Application-Specific Integrated Circuit (ASIC) or, as in our case, for prototyping on a Field Programmable

Table 4 Quantitative results for images (in terms of PSNR, dB) corrupted with 10 % noise density impulse noise.

Method	Images				
	Lena	Goldhill	Elaine	Bridge	Aerial
ACWM	33.79	30.90	32.24	26.17	27.33
ATMBM	36.66	33.64	37.41	27.76	28.48
DRID	34.62	31.27	32.66	26.68	27.99
NAVF	30.87	28.63	31.48	23.96	23.92
LCNR	36.94	33.88	37.35	27.91	29.17
DTBDM	37.15	34.10	36.51	27.77	29.03
Proposed	37.89	35.20	37.72	29.10	29.91

Table 6 Quantitative results for images (in terms of PSNR, dB) corrupted With 20 % noise density impulse noise.

Method	Images				
	Lena	Goldhill	Elaine	Bridge	Aerial
ACWM	31.35	29.39	30.82	25.02	25.69
ATMBM	32.76	30.92	33.38	25.94	26.49
DRID	31.42	29.37	30.61	25.43	26.03
NAVF	30.09	28.17	30.95	23.53	23.56
LCNR	32.80	30.78	32.85	26.13	26.84
DTBDM	31.86	30.39	31.93	25.62	26.37
Proposed	33.48	31.72	33.66	27.19	27.65

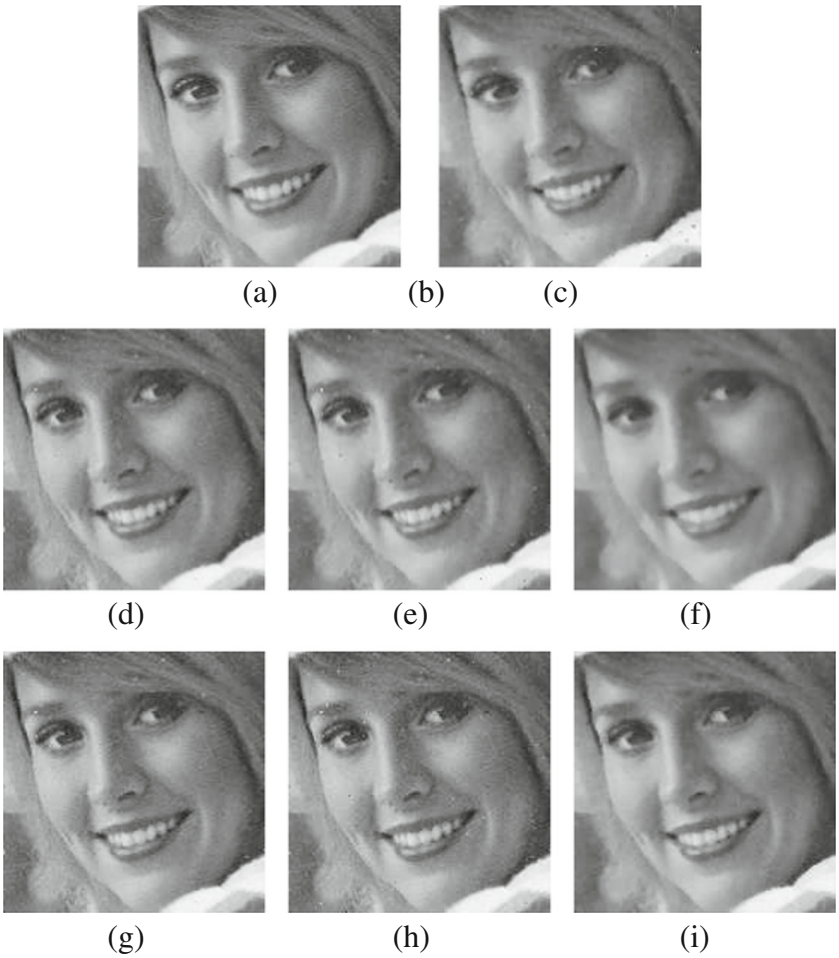


Figure 9 Simulation results of various methods for cropped Elaine image, corrupted with 20 % impulse noise density. (**a**) Original image. (**b**) Noisy image. (**c**) Output of ACWM. (**d**) Output of ATMBM. (**e**) Output of DRID. (**f**) Output of NAVF. (**g**) Output of LCNR. (**h**) Output of DTBDM. (**i**) Output obtained from the proposed method.

Gate Array (FPGA). Whereas the typically sequential software execution model greatly benefits from the high clock frequencies of general purpose Central Processing Units (CPUs), hardware execution must exploit parallelism contained in the program to achieve an adequate execution speed and allow lower clock frequencies. Low and medium level image processing algorithms are often composed of loop programs, where the same instructions will be executed over and over again for each pixel of the image. Pipelining and overlapping of the execution of these instructions is usually a first step towards high throughput implementations. Furthermore, in the same way as caching can significantly reduce the effects of the memory bottleneck on CPU-based architectures, data streaming using First In, First Out (FIFO) interfaces is a means to considerably reduce the amount of cycles to fetch new pixels in comparison to reading from and writing to Dual-Ported Random Access Memory (DPRAM) on FPGAs. To exploit such a hardware-specific implementation, PARO synthesizes accelerators using the data streaming paradigm, which in addition allows us to exploit temporal parallelism for pipelining. The actual accelerator is synthesized as a Finite State Machine with Data Path

Table 7 Quantitative comparison (in terms of PSNR, dB) of (the performance of proposed noisy pixel detection algorithm with) edge preserving filter and standard median filter.

Noise	Method	Image				
density	Lena	Goldhill	Elaine	Bridge	Aerial	
10 %	Proposed approach with median filter	35.88	34.35	37.26	28.17	28.42
	Proposed approach with edge preserving filter	37.89	35.20	37.72	29.10	29.91
20 %	Proposed approach with median filter	32.51	31.40	33.42	26.63	26.96
	Proposed approach with edge preserving filter	33.48	31.72	33.66	27.19	27.65

Table 8 Operator requirements and resulting latency for high-level synthesis of the proposed denoising algorithm with PARO at varying *II*.

II	ADD	SUB	MUL	DIV	ALU	SHIFT	LAT
1	86	28	81	9	401	10	167
2	43	14	41	5	201	5	168
4	22	7	21	3	101	3	174
8	11	4	11	2	51	2	180
16	6	2	6	1	26	1	197

(FSMD), which is the same hardware architecture as used for hand crafted implementations at Register Transfer Level (RTL). The data path is synthesized in the form of a Processing Element (PE), which contains the functional units, as well as an appropriate memory architecture. For processing an image with a sliding window based operator, PARO evaluates the memory access pattern and synthesizes a line buffer to temporarily store pixels for repeated access. As such buffers might become very large for high resolution images, they are implemented using DPRAM on the FPGA. For the actual processing, the pixels stored in the line buffer are extracted into a register-based memory window to enable arbitrary access. Moreover, PARO also generates the control path and the accelerator interface, which uses FIFO semantics. A more detailed description of the hardware architecture synthesized by PARO can be found in [26]. PARO allows to specify the desired throughput of the filter accelerator in terms of the iteration interval (*II*). For an *II* of 1 clock cycle per pixel, the algorithm specification must be completely unrolled, which results in the maximum amount of required resources. Increasing the amount of clock cycles per pixel allows for operators to be reused, which increases

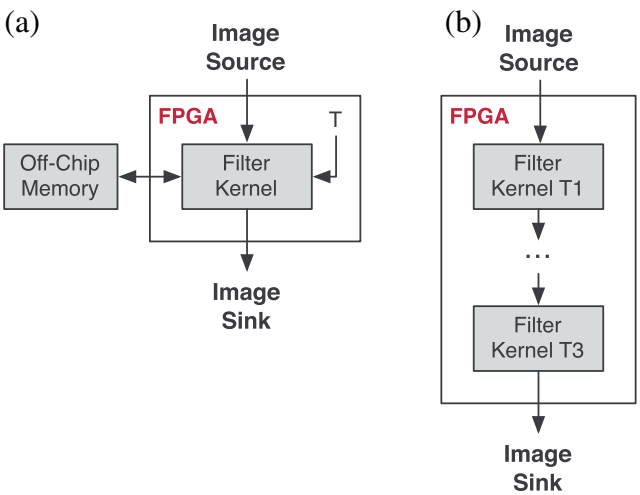


Figure 10 Schematic representation of (**a**) iterative and (**b**) pipelined implementation, where T1 to T3 represent successive applications of different threshold values.

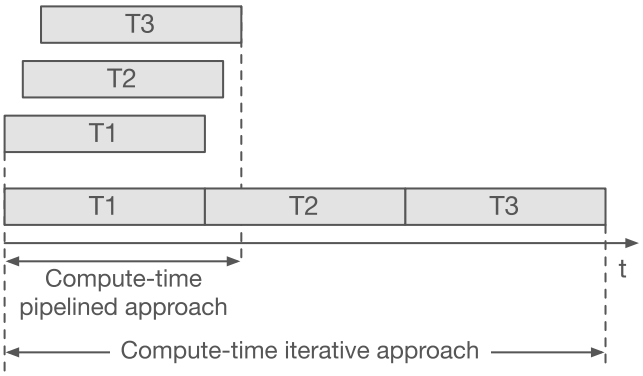


Figure 11 Conceptual comparison between the computed-times of the pipelined and the iterative approach, where T1 to T3 represent successive applications of different threshold values.

the latency per pixel (LAT) but also decreases the amount of required operators.

Table 8 lists the amount of required operators for synthesis of a single filter stage at different values for the II, and specifies the resulting latency per input value. Required operators include standard arithmetic functions, such as addition (ADD), subtraction (SUB), multiplication (MUL), and division (DIV). Moreover, the algorithm makes use of comparison and logic functions, which are provided by an arithmetic-logic-unit (ALU), and also requires bit-shift operations (SHIFT). In terms of operators, PARO allows to use custom implementations. To enable scheduling, only the latency and iteration interval of each operator must be provided to the tool.

According to the amount of available resources, the algorithm can either be realized as an iterative or a pipelined approach, which is illustrated in Fig. 10. For the iterative approach, the filter design is instantiated only once in hardware and reused to apply a different threshold at each run. In this scenario, the latency LAT_{it} of the denoising algorithm in clock cycles can be obtained as

$$LAT_{it} = (II * dim_x * dim_y + LAT) * K, \tag{14}$$

where the image dimensions are defined by dim_x and dim_y, and K is the number of successive applications of different thresholds.

One disadvantage of the iterative approach is the very high processing duration, as successor iterations may only be started after the predecessor has finished processing the

Table 9 Available resources for differently sized Xilinx FPGAs.

FPGA	Model Name	Slices	FF	LUT	BRAM
Artix 7	XC7A200T	33,650	269,200	215,360	365
Kintex 7	XC7K325T	50,950	407,600	326,080	445
Virtex 7	XC7VX485T	75,900	607,200	485,760	1030

Table 10 PPNR implementation results on differently sized FPGAs given in percentage of overall available resources.

FPGA	II	LUT	FF	RAM	F [MHz]	P [W]	L [ms]
Artix 7	1	47.57	28.21	2.74	148.79	0.88	1.77
	2	47.25	20.23	2.74	129.62	0.51	4.05
	4	33.18	16.67	2.74	121.61	0.41	8.62
	8	30.72	11.40	2.74	108.38	0.34	19.36
	16	29.19	8.68	2.74	104.54	0.38	40.13
Kintex 7	1	32.25	18.45	2.25	222.77	0.93	1.18
	2	32.30	13.25	2.25	203.83	0.77	2.57
	4	23.03	10.89	2.25	192.98	0.54	5.44
	8	22.22	7.49	2.25	169.55	0.48	12.37
	16	20.39	5.70	2.25	167.11	0.39	25.10
Virtex 7	1	21.65	12.39	0.97	214.96	1.03	1.22
	2	21.72	8.90	0.97	189.61	0.88	2.77
	4	15.47	7.32	0.97	183.39	0.66	5.72
	8	14.01	5.02	0.97	163.45	0.58	12.83
	16	13.69	3.82	0.97	161.84	0.49	25.92

image in its entirety. Furthermore, the image data must be either retained on chip, which is often not possible due to insufficient memory resources on FPGAs, or kept in off-chip memory (see Fig. 10a), which would cause increased latency. In contrast to the iterative approach, it is desirable to make use of the massive parallelism of the FPGAs as much as possible, which can be achieved by constructing the application of successive threshold values as a denoising pipeline (see Fig. 10b). Using this approach, the successor operation can be started as soon as the predecessor presents the first pixel. As the iteration latency of the filter implementation is typically several orders of magnitude lower than the compute-time for the entire image, applying different threshold values can be computed almost fully parallel. A conceptual illustration of this principle is shown in Fig. 11. The overall latency LAT_{pipe} in clock cycles is in this case

$$LAT_{pipe} = (II * dim_x * dim_y) + (K * LAT). \quad (15)$$

Moreover, both approaches can also be combined in order to adjust the implementation to a wide range of design goals. Using a high-level synthesis tool like PARO for design conception does not only aim at increasing productivity during design time, it also shows its advantage for implementation, as a once developed filter design can be easily adapted to contrasting design goals and synthesizable hardware modules can be generated in a very short time to facilitate efficient design space exploration. PARO is capable of generating hardware module implementations according to the amount of available resources in terms of arithmetic operators, as well as design throughput in terms of the iteration interval. Aside from the minimum clock period at which the design can be implemented on the target architecture, this represents the most influential design criteria with respect to the processing time. If a dedicated resource can be instantiated for every operation of the algorithm, the design can process a new pixel at every clock cycle, yielding the fastest but also most expensive design. If the design goals can be relaxed in order to allow for multiple clock cycles per pixel, arithmetic operators can be shared to decrease the logic complexity.

For this study, we have used PARO to create synthesizable VHDL descriptions of the denoising algorithm for images of size 512 × 512 pixels at various iteration intervals to obtain differently sized designs for implementation on FPGAs. Furthermore, to demonstrate the algorithm's applicability for realization on hardware, we show implementation results for differently sized FPGAs from Xilinx, which represent different target options with respect to cost and energy consumption. The Post Place and Route (PPNR) resource and performance characteristics were obtained by synthesizing the accelerators in Xilinx Vivado as

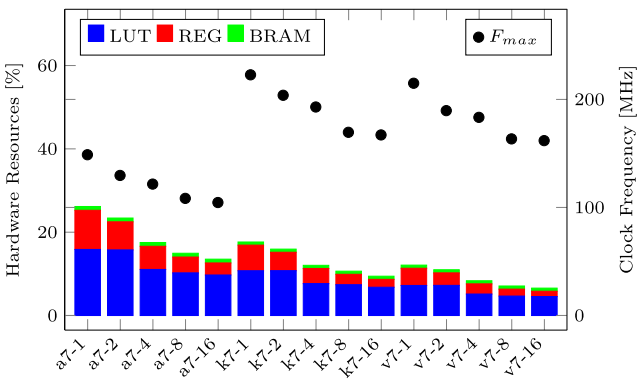


Figure 12 Stacked representation of the amount of FPGA resources.

Table 11 PPNR results of LCNR, DTBDM, and the here proposed approach on the XC7K325T FPGA.

Design	II	LUT (%)	FF (%)	RAM (%)	F [MHz]	P [W]	L [ms]
LCNR	1	4.18	1.53	0.1	234.30	0.202	1.12
DTBDM	1	3.02	1.43	0.1	245.45	0.198	1.07
Proposed	1	32.25	18.45	2.25	222.77	0.928	1.18

out-of-context modules. To obtain the power requirements, we have created PPNR simulation netlists and simulated these for various input images using Mentor Graphics QuestaSim to record the switching activity and obtain power estimates using Vivado. The available hardware resources for the evaluated FPGAs are given in Table 9. The Artix 7 FPGA is the smallest and most power efficient, whereas the Virtex 7 provides the most resources, however, it also requires the most power. The Kintex 7 is situated in the middle between the Artix 7 and the Virtex 7 in terms of resources and power consumption. Table 10 presents PPNR results for Artix 7, Kintex 7, and Virtex 7, including the required amount of lookup tables (LUT), flipflops (FF), and block RAM (RAM), given as percentage of overall available resources of that type. Furthermore, the table shows the maximum achievable clock frequency (F) in MHz and power requirements (P) in Watt for varying iteration intervals (II). In addition, the table lists the latency (L) for processing one entire image of 512 × 512 16-bit pixels in milliseconds.

FPGAs have meanwhile become very complex, so that the most resource hungry design at iteration interval 1 does not even overwhelm the smallest chosen FPGA. Increasing the iteration interval effectively lowers the resource and power requirements of the design. Due to the increased routing complexity for operator reuse, however, the maximum clock frequency must also be lowered. Figure 12 compares the different solutions in terms of total FPGA resource requirements, given as percentage of overall available resources, and maximum achievable clock frequency.

It can be seen that the designs require well below 40 % of total FPGA resources, leaving enough space for additional logic for extra tasks or integration of the design.

It is noteworthy that, since we assume data streaming, the implementation is highly scalable and can be adapted to higher image resolutions with only marginal increase in hardware cost. However, increasing the image resolution would cause a longer processing time.

5.3 Comparison

In terms of image quality, the most challenging competitors with the here presented approach are LCNR [23] and DTBDM [20]. To allow for a fair comparison, we have implemented the two algorithms in VHDL according to the details provided in the respective publications and evaluated the algorithms with respect to performance and resource requirements on the Xilinx Kintex 7 XC7K325T FPGA. Table 11 lists the PPNR implementation results. As the here presented approach uses three kernel iterations, it comes at no surprise that the hardware implementation of our algorithm requires substantially more resources than the other two approaches. Another factor that contributes to the increased resource usage is that we actually need to use real dividers for the implementation of the divisions (refer to Equations (6) and (11)), whereas DTBDM, for example, can implement the divisions using shift operators. In contrast, the execution speed of our algorithm is almost identical to the other two algorithms while delivering superior image quality.

6 Conclusion

In this paper, we have presented an efficient noise reduction algorithm capable of restoring the images corrupted with random-valued impulse noise. The proposed scheme operates in two phases, namely the impulse detection phase with a follow-up filtering operation on the detected noisy pixels in the second stage. The impulse detection scheme consists of a novel sliding mean vector approach in which the mean of the neighboring pixels is analyzed to determine the nature of the central pixel in the candidate window. The qualitative and quantitative results illustrate the superior image quality of the proposed algorithm over existing noise reduction techniques. Since noise reduction operation is often present in consumer electronic devices we have also developed a deeply pipelined parallel hardware architecture for our proposed algorithm using the HLS tool PARO. Although the above presented implementation indicates that the algorithm has a significantly higher resource usage than other techniques it can deliver a superior image quality at almost the same execution speed.

Acknowledgments This work was supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center “Invasive Computing” (SFB/TR 89).

References

1. The Concise Encyclopedia of Statistics. Springer, New York. (2008).

2. Aizenberg, I., & Butakoff, C. (2004). Effective impulse detector based on rank-order criteria. *IEEE Signal Processing Letters*, *11*(3), 363–366. doi:10.1109/LSP.2003.822925.
3. Altera Corp (2013). Altera SDK for OpenCL Programming Guide.
4. Arvind, N.R. (2008). Hands-on Introduction to Bluespec System Verilog (BSV). In *6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design, 2008. MEMOCODE 2008, pp 205–206*. doi:10.1109/MEMCOD.2008.4547713.
5. Bhadouria V.S., & Ghoshal D. (2015). A study on genetic expression programming-based approach for impulse noise reduction in images. Signal, Image and Video Processing pp 1–10, doi:10.1007/s11760-015-0780-6.
6. Bhadouria, V.S., Ghoshal, D., & Siddiqi, A.H. (2014). A new approach for high density saturated impulse noise removal using decision-based coupled window median filter. *Signal, Image and Video Processing*, *8*(1), 71–84.
7. Calypto Design Systems Inc (2012). Calypto Product Family Datasheet.
8. Chen, T., & Wu, H.R. (2001). Adaptive impulse detection using center-weighted median filters. *IEEE Signal Processing Letters*, *8*(1), 1–3.
9. Dong, Y., & Xu, S. (2007). A new directional weighted median filter for removal of random-valued impulse noise. *IEEE Signal Processing Letters*, *14*(3), 193–196. doi:10.1109/LSP.2006.884014.
10. Eng, H.L., & Ma, K.K. (2001). Noise adaptive soft-switching median filter. *IEEE Transactions on Image Processing*, *10*(2), 242–251. doi:10.1109/83.902289.
11. Esakkirajan S., Veerakumar T., Subramanyam A., & Prem-Chand C. (2011). Removal of high density salt and pepper noise through modified decision based unsymmetric trimmed median filter. *IEEE Signal Processing Letters*, *18*(5), 287–290. doi:10.1109/LSP.2011.2122333.
12. Feautrier P., & Lengauer C. (2011). Polyhedron model. In *Padua DA (ed) Encyclopedia of Parallel Computing, Springer, pp 1581–1592*. doi:10.1007/978-0-387-09766-4_502.
13. Fischer, V., Lukac, R., & Martin, K. (2005). Cost-effective video filtering solution for real-time vision systems. *EURASIP Journal on Applied Signal Processing*, *2005*, 2026–2042.
14. Gonzalez, R.C., & Woods, R.E. (2002). *Digital image processing*. Engle-wood Cliffs: Prentice-Hall.
15. Gupta, S., Dutt, N., Gupta, R., & Nicolau, A. (2003). SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations. In *Proceedings of the 16th International Conference on VLSI Design, pp 461–466*.
16. Hannig, F. (2009). Scheduling techniques for high-throughput loop accelerators. Dissertation, University of Erlangen-Nuremberg, Germany, Verlag Dr Hut, Munich, Germany.
17. Hannig, F., Ruckdeschel, H., Dutta, H., & Teich, J. (2008). In *PARO: Synthesis of hardware accelerators for multi-dimensional dataflow-intensive applications Proceedings of the Fourth International Workshop on Applied Reconfigurable Computing (ARC), Springer, Lecture Notes in Computer Science (LNCS), vol 4943, pp 287–293*. doi:10.1007/978-3-540-78610-8_30.
18. Huang, T., Yang, G., & Tang, G. (1979). A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *27*(1), 13–18. doi:10.1109/TASSP.1979.1163188.
19. Hwang H., & Haddad R. (1995). Adaptive median filters: new algorithms and results. *IEEE Transactions on Image Processing*, *4*(4), 499–502. doi:10.1109/83.370679.
20. Lien, C.Y., Huang, C.C., Chen, P.Y., & Lin, Y.F. (2013). An efficient denoising architecture for removal of impulse noise in images. *IEEE Transactions on Computers*, *62*(4), 631–643. doi:10.1109/TC.2011.256.
21. Luo, W. (2006). An efficient detail-preserving approach for removing impulse noise in images. *IEEE Signal Processing Letters*, *13*(7), 413–416. doi:10.1109/LSP.2006.873144.
22. Ma, Z., He, K., Wei, Y., Sun, J., & Wu, E. (2013). Constant time weighted median filtering for stereo matching and beyond. In *Computer vision (ICCV), 2013 IEEE International Conference on, IEEE, pp 49-56*.
23. Matsubara T., Moshnyaga V.G., & Hashimoto K. (2010). A fpga implementation of low-complexity noise removal. In *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2010, IEEE, pp 255–258*.
24. Petrovic, N., & Crnojevic, V. (2008). Universal impulse noise filter based on genetic programming. *IEEE Transactions on Image Processing*, *17*(7). doi:10.1109/TIP.2008.924388.
25. Saeedi, J., Moradi, M.H., & Faez, K. (2010). A new wavelet-based fuzzy single and multi-channel image denoising. *Image and Vision Computing*, *28*(12), 1611–1623. doi:10.1016/j.imavis.2010.04.004.
26. Schmid, M., Hannig, F., Tanase, A., & Teich, J. (2014). High-level synthesis revised – Generation of FPGA accelerators from a domain-specific language using the polyhedron model. In *Parallel Computing: Accelerating Computational Science and Engineering (CSE), Advances in Parallel Computing, vol 25, IOS Press, Amsterdam, The Netherlands, pp 497–506*. doi:10.3233/978-1-61499-381-0-497.
27. Schulte, S., De Witte, V., Nachtegael, M., Van der Weken, D., & Kerre, E. (2006). Fuzzy two-step filter for impulse noise reduction from color images. *IEEE Transactions on Image Processing*, *15*(11), 3567–3578. doi:10.1109/TIP.2006.877494.
28. Schulte, S., Nachtegael, M., De Witte, V., Van der Weken, D., & Kerre, E. (2006). A fuzzy impulse noise detection and reduction method. *IEEE Transactions on Image Processing*, *15*(5), 1153–1162. doi:10.1109/TIP.2005.864179.
29. Toh, K., & Isa, N. (2010). Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction. *IEEE Signal Processing Letters*, *17*(3), 281–284. doi:10.1109/LSP.2009.2038769.
30. Villarreal, J., Park, A., Najjar, W., & Halstead, R. (2010). Designing Modular Hardware Accelerators in C with ROCCC 2.0. *Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, *0*, 127–134. doi:10.1109/FCCM.2010.28.
31. Xilinx Inc (2013). Vivado Design Suite User Guide - High-Level synthesis.
32. Xiong, B., & Yin, Z. (2012). A universal denoising framework with a new impulse detector and nonlocal means. *IEEE Transactions on Image Processing*, *21*(4), 1663–1675. doi:10.1109/TIP.2011.2172804.
33. Xue, J. (1997). Unimodular transformations of non-perfectly nested loops. *Parallel Computing*, *22*, 1621–1645.
34. Yagou, H., Ohtake, Y., & Belyaev, A. (2002). Mesh smoothing via mean and median filtering applied to face normals. In *Geometric Modeling and Processing, 2002. Proceedings, pp 124–131*. doi:10.1109/GMAP.2002.1027503.
35. Yang, Q., Ahuja, N., & Tan, K.H. (2014). Constant time median and bilateral filtering. *International Journal of Computer Vision*, 1–12.
36. Yu, H., Zhao, L., & Wang, H. (2008). An efficient procedure for removing random-valued impulse noise in images. *IEEE Signal Processing Letters*, *15*, 922–925. doi:10.1109/LSP.2008.2005051.
37. Zhang, Q., Xu, L., & Jia, J. (2014). 100+ times faster weighted median filter (wmf). In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pp 2830–2837*. doi:10.1109/CVPR.2014.362.

Vivek Singh Bhadouria received his Bachelor of Technology in Electronics & Communication Engineering from Gautum Buddha Technical University, Lucknow, India in 2010. He received his PhD in Electronics & Communication Engineering from National Institute of Technology Agartala, India in 2015. His areas of interest are computer vision, image processing, and machine learning.

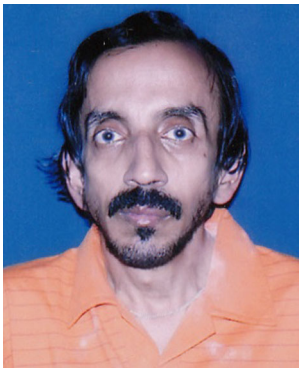
Alexandru Tanase is a doctoral researcher at the Computer Science Department of Friedrich-Alexander University Erlangen-Nürnberg (FAU), since 2011. He received his diploma degree in computer engineering in 2006 and master degree in parallel processing in 2008 from ULBS University, Romania. His main research interests include high-level synthesis, programmable hardware accelerators, the design of massively parallel architectures, mapping methodologies for domain-specific computing, and architecture/compiler co-design.

Moritz Schmid received a Ph.D. degree in CS from the Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany in 2015. Afterwards, he joined R&D of the Advanced Therapies business unit at Siemens Healthcare GmbH, Germany. Moritz' research interests include system-level design automation for FPGA-based SoCs and acceleration of medical imaging on heterogeneous embedded systems.

Frank Hannig leads the Architecture and Compiler Design Group in the CS Department at Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany, since 2004. He received a diploma degree in an interdisciplinary course of study in EE and CS from University of Paderborn, Germany in 2000 and a Ph.D. degree (Dr.-Ing.) in CS from FAU in 2009. His main research interests are the design of massively parallel architectures, ranging from dedicated hardware to multi-core architectures, mapping methodologies for domain-specific computing, and architecture/compiler co-design. Frank has authored or co-authored more than 150 peer-reviewed publications. He serves on the program committees of several international conferences (ARC, ASAP, CODES+ISSS, DATE, DASIP, SAC). Frank is a senior member of the IEEE and an affiliate member of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC).

Jürgen Teich is with Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany, where he is directing the Chair for Hardware/Software Co-Design since 2003. He received the M.S. degree (Dipl.-Ing.; with honors) from the University of Kaiserslautern, Germany in 1989 and the Ph.D. degree (Dr.-Ing.; summa cum laude) from the University of Saarland, Saarbruecken, Germany, in 1993. In 1994, he joined the DSP design group of Prof. E. A. Lee in the Department of Electrical Engineering and Computer Sciences (EECS), University of California at Berkeley (PostDoc). From 1995 to 1998, he held a position at the Institute of Computer Engineering and Communications Networks Laboratory (TIK), ETH Zurich, Switzerland with his Habilitation on the topic of 'Synthesis and Optimization of Digital Hardware/Software Systems' in 1996. From 1998 to 2002, he was Full Professor in the Electrical Engineering and Information Technology Department, University of Paderborn, Germany.

Prof. Teich is involved in many interdisciplinary projects on basic research as well as industrial projects. From 2003-2009, he was an elected board member (Fachkollegiat) of the Deutsche Forschungsgemeinschaft (DFG) for the area of Computer Architecture and Embedded Systems. He has been the initiator and coordinator of the DFG priority programme 1148 on "Reconfigurable Computing". Since 2010, he has also been the principal coordinator of the Transregional Research Center 89 "Invasive Computing" funded by the German Research Foundation (DFG). In 2011, he was elected member of the Academia Europaea.



Dibyendu Ghoshal has received his B.Sc. (Honours in Physics), B.Tech & M.Tech in Radiophysics and Electronics, all from Calcutta University (CU) in 1981, 1985 and 1987 respectively. He joined Indian Engineering Sevices (Group-A) in 1988 and served Department of Telecommunication, GOI. After holding various posts for three years, D. Ghoshal left the job to pursue higher studies and was awarded SRF in 1992 by CSIR. Subsequently, he was awarded Postdoctoral Research Associateship in 1996 and PhD in Radiophysics & Electronics from CU in 1997 with specialization in Microwave and millimeterwave systems. D. Ghoshal has served as post doctoral research associate and scientist for nearly 12 years and was associated in various projects sponsored by GOI. His research interest includes micro & millimeter wave, semiconductor physics & devices, Digital Signal Processing and Image Processing.