An Efficient Denoising Architecture for Removal of Impulse Noise in Images

Chih-Yuan Lien, Chien-Chuan Huang, Pei-Yin Chen, *Member*, *IEEE*, and Yi-Fan Lin

Abstract—Images are often corrupted by impulse noise in the procedures of image acquisition and transmission. In this paper, we propose an efficient denoising scheme and its VLSI architecture for the removal of random-valued impulse noise. To achieve the goal of low cost, a low-complexity VLSI architecture is proposed. We employ a decision-tree-based impulse noise detector to detect the noisy pixels, and an edge-preserving filter to reconstruct the intensity values of noisy pixels. Furthermore, an adaptive technology is used to enhance the effects of removal of impulse noise. Our extensive experimental results demonstrate that the proposed technique can obtain better performances in terms of both quantitative evaluation and visual quality than the previous lower complexity methods. Moreover, the performance can be comparable to the higher complexity methods. The VLSI architecture of our design yields a processing rate of about 200 MHz by using TSMC 0.18 μm technology. Compared with the state-of-the-art techniques, this work can reduce memory storage by more than 99 percent. The design requires only low computational complexity and two line memory buffers. Its hardware cost is low and suitable to be applied to many real-time applications.

Index Terms—Image denoising, impulse noise, impulse detector, architecture

1 Introduction

Image processing is widely used in many fields, such as medical imaging, scanning techniques, printing skills, license plate recognition, face recognition, and so on. In general, images are often corrupted by impulse noise in the procedures of image acquisition and transmission. The noise may seriously affect the performance of image processing techniques. Hence, an efficient denoising technique becomes a very important issue in image processing [1], [2]. According to the distribution of noisy pixel values, impulse noise can be classified into two categories: fixed-valued impulse noise and random-valued impulse noise. The former is also known as salt-and-pepper noise because the pixel value of a noisy pixel is either minimum or maximum value in gray-scale images. The values of noisy pixels corrupted by random-valued impulse noise are uniformly distributed in the range of [0, 255] for gray-scale images. There have been many methods for removing salt-and-pepper noise, and some of them perform very well [3], [4], [5], [6], [7]. The random-valued impulse noise is more difficult to handle due to the random distribution of noisy pixel values. We only focus on removing the random-valued impulse noise from the corrupted image in this paper.

- *C.-Y. Lien is with the Department of Electronic Engineering, National Kaohsiung University of Applied Sciences, Kaohsiung, Taiwan, R.O.C. E-mail: cylien@cc.kuas.edu.tw.*
- *C.-C. Huang, P.-Y. Chen, and Y.-F. Lin are with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Road, Tainan 70101, Taiwan, R.O.C. E-mail: chien.chuan.huang@gmail.com, pychen@csie.ncku.edu.tw, ste1028@hotmail.com.*

Manuscript received 7 June 2011; revised 26 Oct. 2011; accepted 20 Dec. 2011; published online 27 Dec. 2011.
Recommended for acceptance by L. Wang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-06-0379.
Digital Object Identifier no. 10.1109/TC.2011.256.

Recently, many image denoising methods have been proposed to carry out impulse noise suppression [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. Some of them employ the standard median filter [8] or its modifications [9], [10]. However, these approaches might blur the image since both noisy and noise-free pixels are modified. To avoid the damage on noise-free pixels, an efficient switching strategy has been proposed in the literature [11], [12], [13]. In general, the switching median filter consists of two steps: 1) impulse detection and 2) noise filtering. It locates the noisy pixels with an impulse detector, and then filters them rather than the whole pixels of an image to avoid causing the damage on noise-free pixels. In addition to median filer, there are other methods used to carry out impulse noise. In [14], Luo proposed an alpha-trimmed mean-based method (ATMBM). It used the alpha-trimmed mean in impulse detection and replaced the noisy pixel value by a linear combination of its original value and the median of its local window. A differential rank impulse detector (DRID) was presented in [15]. The impulse detector of DRID is based on a comparison of signal samples within a narrow rank window by both rank and absolute value. In [16], Yu et al. proposed a method using a statistic of rank-ordered relative differences (RORD-WMF) to identify pixels which are likely to be corrupted by impulse noise. A directional weighted median (DWM) method proposed by Dong and Xu was presented in [17]. It is based on the differences between the current pixel and its neighbors aligned with four main directions. In [18], Petrović and Crnojević proposed a method that employed genetic programming for impulse noise filter construction. The method is based on the switching scheme with cascaded detectors and corresponding estimators.

Generally, the denoising methods can be classified into two categories: lower complexity techniques [8], [9], [10], [11], [12], [13] and higher complexity techniques [14], [15], [16], [17], [18]. The complexity of denoising algorithms

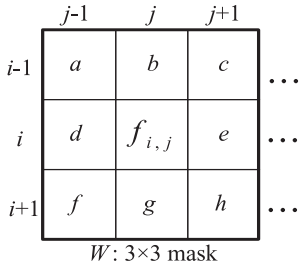


Fig. 1. A 3×3 mask centered on $p_{i,j}$.

depends mainly on the local window size, memory buffer, and iteration times. The lower complexity techniques use a fixed-size local window, require a few line buffers, and perform no iterations. Therefore, the computational complexity is low. However, the reconstructed image quality is not good enough. The higher complexity techniques yield visually pleasing images by using high computational complexity arithmetic operations, enlarging local window size adaptively or doing iterations. The higher complexity approaches require long computational time as well as full frame buffer. Today, in many practical real-time applications, the denoising process is included in end-user equipment, so there appears an increasing need of a good lower-complexity denoising technique, which is simple and suitable for low-cost VLSI implementation. Low cost is a very important consideration in purchasing consumer electronic products. To achieve the goal of low cost, less memory and easier computations are indispensable. In this paper, we focus only on the lower complexity denoising techniques because of its simplicity and easy implementation with the VLSI circuit.

The decision tree is a simple but powerful form of multiple variable analysis [19]. It can break down a complex decision-making process into a collection of simpler decisions, thus provide a solution which is often easier to interpret [20]. There have been several methods using decision tree to deal with salt-and-pepper noise [4], [5], [21], [22], [23], [37] and some of them perform well.

Based on above basic concepts, we present a novel adaptive decision-tree-based denoising method (DTBDM) and its VLSI architecture for removing random-valued impulse noise. To enhance the effects of removal of impulse noise, the results of reconstructed pixels are adaptively written back as a part of input data. The proposed design requires simple computations and two line memory buffers only, so its hardware cost is low. For a 512×512 8-bit gray-scale test image, only two line buffer $(512 \times 2 \times 8 \text{ bits})$ is needed in our design. Most state-of-the-art methods need to buffer a full image $(512 \times 512 \times 8 \text{ bits})$. In our design, 99.6 percent of storage is reduced. Furthermore, only simple arithmetic operations, such as addition and subtraction, are used in DTBDM. Especially, it can remove the noise from corrupted images efficiently and requires no previous training. Our extensive experimental results demonstrate that the proposed technique can obtain better performances in terms of both quantitative evaluation and visual quality than other lower complexity denoising methods [8], [9], [10], [11], [12], [13]. Moreover, the performance can be comparable to the higher complexity methods [14], [15], [16]. The seven-stage VLSI architecture for the proposed design was implemented and synthesized by using Verilog HDL and Synopsys Design Compiler, respectively. In our simulation, the circuit can achieve 200 MHz with only 21k gate counts by using TSMC $0.18\ \mu\text{m}$ technology.

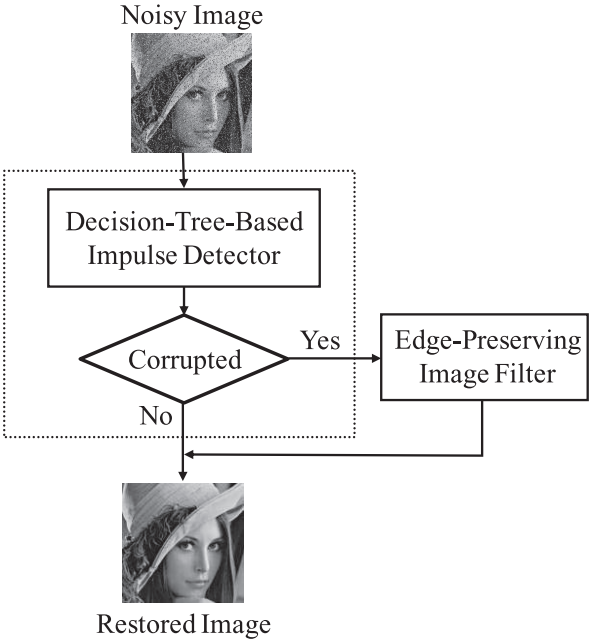


Fig. 2. The dataflow of DTBDM.

The rest of this paper is organized as follows. The proposed DTBDM is introduced briefly in Section 2. Section 3 describes the proposed VLSI architecture in detail. Section 4 illustrates the VLSI implementation and comparisons. The conclusion is provided in Section 5.

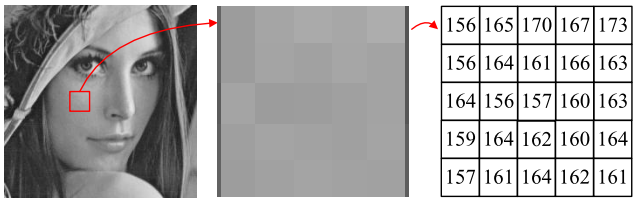
2 The Proposed DTBDM

The noise considered in this paper is random-valued impulse noise with uniform distribution as practiced in [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. Here, we adopt a 3×3 mask for image denoising. Assume the pixel to be denoised is located at coordinate (i, j) and denoted as $p_{i,j}$, and its luminance value is named as $f_{i,j}$, as shown in Fig. 1. According to the input sequence of image denoising process, we can divide other eight pixel values into two sets: $W_{TopHalf}$ and $W_{BottomHalf}$. They are given as

$$W_{TopHalf} = \{a, b, c, d\}. \quad (1)$$

$$W_{BottomHalf} = \{e, f, g, h\}. \quad (2)$$

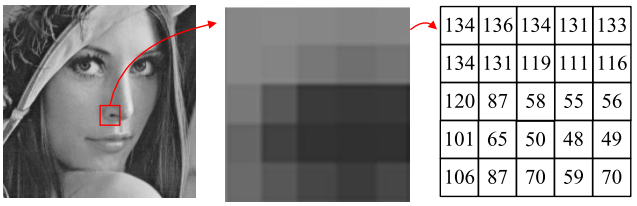
DTBDM consists of two components: decision-tree-based impulse detector and edge-preserving image filter. The detector determines whether $p_{i,j}$ is a noisy pixel by using the decision tree and the correlation between pixel $p_{i,j}$ and its neighboring pixels. If the result is positive, edge-preserving image filter based on direction-oriented filter generates the reconstructed value. Otherwise, the value will be kept unchanged. The design concept of the DTBDM is displayed in Fig. 2.



156	165	170	167	173
156	164	161	166	163
164	156	157	160	163
159	164	162	160	164
157	161	164	162	161

(a) original image (b) Region of red rectangle (c) Gray-scale value

Fig. 3. A smooth region in Lena.



134	136	134	131	133
134	131	119	111	116
120	87	58	55	56
101	65	50	48	49
106	87	70	59	70

(a) original image (b) Region of red rectangle (c) Gray-scale value

Fig. 4. A nonsmooth region in Lena.

2.1 Decision-Tree-Based Impulse Detector

In order to determine whether $p_{i,j}$ is a noisy pixel, the correlations between $p_{i,j}$ and its neighboring pixels are considered [10], [11], [14], [16], [17], [23], [24], [25], [26], [27], [28], [29], [30]. Surveying these methods, we can simply classify them into several ways—observing the degree of isolation at current pixel [10], [11], [14], [16], [17], [23], [24], [25], determining whether the current pixel is on a fringe [16], [17], [26], [27] or comparing the similarity between current pixel and its neighboring pixels [16], [28], [29], [30]. Therefore, in our decision-tree-based impulse detector, we design three modules—isolation module (IM), fringe module (FM), and similarity module (SM). Three concatenating decisions of these modules build a decision tree. The decision tree is a binary tree and can determine the status of $p_{i,j}$ by using the different equations in different modules. First, we use isolation module to decide whether the pixel value is in a smooth region. If the result is negative, we conclude that the current pixel belongs to noisy free. Otherwise, if the result is positive, it means that the current pixel might be a noisy pixel or just situated on an edge. The fringe module is used to confirm the result. If the current pixel is situated on an edge, the result of fringe module will be negative (noisy free); otherwise, the result will be positive. If isolation module and fringe module cannot determine whether current pixel belongs to noisy free, the similarity module is used to decide the result. It compares the similarity between current pixel and its neighboring pixels. If the result is positive, $p_{i,j}$ is a noisy pixel; otherwise, it is noise free. The following sections describe the three modules in detail.

2.1.1 Isolation Module

The pixel values in a smooth region should be close or locally slightly varying, as shown in Fig. 3. The differences between its neighboring pixel values are small. If there are noisy values, edges, or blocks in this region, the distribution of the values is different, as shown in Fig. 4. Therefore, we determine whether current pixel is an isolation point by observing the smoothness of its surrounding pixels. Fig. 5 shows an example of noisy image. The pixels with shadow suffering from noise have low similarity with the neighboring pixels and the so-called isolation point. The difference between it and its neighboring pixel value is large. According to the above concepts, we first detect the maximum and minimum luminance values in $W_{TopHalf}$, named as *TopHalf_max*, *TopHalf_min*, and calculate the difference between them, named as *TopHalf_diff*. For $W_{BottomHalf}$, we apply the same idea to obtain *BottomHalf_diff*. The two difference values are compared with a threshold Th_IM_a to decide whether the surrounding region belongs to a smooth area. The equations are as

$$TopHalf\_diff = TopHalf\_max - TopHalf\_min. \tag{3}$$

$$BottomHalf\_diff = BottomHalf\_max - BottomHalf\_min. \tag{4}$$

$$DecisionI = \begin{cases} true, & \text{if } (TopHalf\_diff \geq Th\_IM_a) \\ & \text{or } (BottomHalf\_diff \geq Th\_IM_a) \\ false, & \text{otherwise.} \end{cases} \tag{5}$$

Next, we take $p_{i,j}$ into consideration. Two values must be calculated first. One is the difference between $f_{i,j}$ and *TopHalf_max*; the other is the difference between $f_{i,j}$ and *TopHalf_min*. After the subtraction, a threshold Th_IM_b is used to compare these two differences. The same method as in the case of $W_{BottomHalf}$ is applied. The equations are as

$$IM\_TopHalf = \begin{cases} true, & \text{if } (|f_{i,j} - TopHalf\_max| \geq Th\_IM_b) \\ & or\ (|f_{i,j} - TopHalf\_min| \geq Th\_IM_b) \\ false, & \text{otherwise.} \end{cases} \tag{6}$$

$$IM\_BottomHalf = \begin{cases} true, & \text{if } (|f_{i,j} - BottomHalf\_max| \geq Th\_IM_b) \\ & or\ (|f_{i,j} - BottomHalf\_min| \geq Th\_IM_b) \\ false, & \text{otherwise.} \end{cases} \tag{7}$$

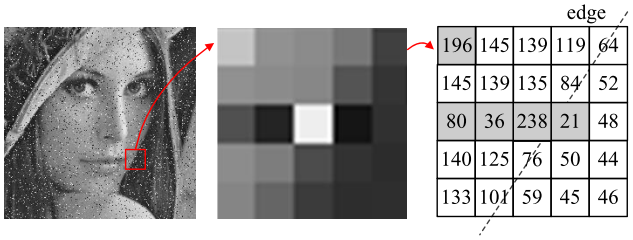
$$DecisionII = \begin{cases} true, & \text{if } (IM\_TopHalf = true) \\ & \text{or } (IM\_BottomHalf = true) \\ false, & \text{otherwise.} \end{cases} \tag{8}$$

Finally, we can make a temporary decision whether $p_{i,j}$ belongs to a suspected noisy pixel or is noisy free.

136	66	145	160	142
134	131	185	141	14
178	111	116	127	127
58	55	56	72	88
50	48	49	63	76

(a) original image (b) Region of red rectangle (c) Gray-scale value

Fig. 5. The difference between noisy and neighboring pixels in Lena.



(a) original image (b) Region of red rectangle (c) Gray-scale value

Fig. 6. The edge region in Lena.

2.1.2 Fringe Module

If $p_{i,j}$ has a great difference with neighboring pixels, it might be a noisy pixel (discussed in Section 2.1.1 IM) or just situated on an edge, as shown in Fig. 6. How to conclude that a pixel is noisy or situated on an edge is difficult. In order to deal with this case, we define four directions, from E_1 to E_4, as shown in Fig. 7. We take direction E_1 for example. By calculating the absolute difference between $f_{i,j}$ and the other two pixel values along the same direction, respectively, we can determine whether there is an edge or not. The detailed equations are as

$$FM\_E_1 = \begin{cases} false, & \text{if } (|a - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|h - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|a - h| \geq Th\_FM_b) \\ true, & \text{otherwise.} \end{cases} \quad (9)$$

$$FM\_E_2 = \begin{cases} false, & \text{if } (|c - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|f - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|c - f| \geq Th\_FM_b) \\ true, & \text{otherwise.} \end{cases} \quad (10)$$

$$FM\_E_3 = \begin{cases} false, & \text{if } (|b - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|g - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|b - g| \geq Th\_FM_b) \\ true, & \text{otherwise.} \end{cases} \quad (11)$$

$$FM\_E_4 = \begin{cases} false, & \text{if } (|d - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|e - f_{i,j}| \geq Th\_FM_a) \\ & \text{or } (|d - e| \geq Th\_FM_b) \\ true, & \text{otherwise.} \end{cases} \quad (12)$$

$$Decision\ III = \begin{cases} false, & \text{if } (FM\_E_1) \text{ or } (FM\_E_2) \\ & \text{or } (FM\_E_3) \text{ or } (FM\_E_4) \\ \text{true}, & \text{otherwise.} \end{cases} \quad (13)$$

2.1.3 Similarity Module

The last module is similarity module. The luminance values in mask W located in a noisy-free area might be close. The median is always located in the center of the variational series, while the impulse is usually located near one of its ends. Hence, if there are extreme big or small values, that implies the possibility of noisy signals. According to this concept, we sort nine values in ascending order and obtain the fourth, fifth, and sixth values which are close to the median in mask W. The fourth, fifth, and sixth values are represented as $4_{th}inW_{i,j}$, $MedianInW_{i,j}$, and $6_{th}inW_{i,j}$. We define $Max_{i,j}$ and $Min_{i,j}$ as

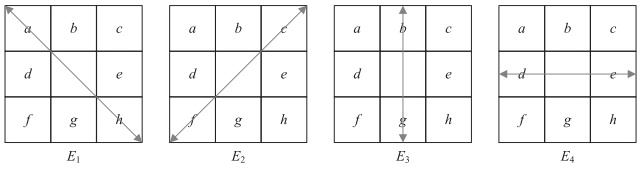


Fig. 7. Four directions in DTBDM.

$$\begin{aligned} Max_{i,j} &= 6_{th}inW_{i,j} + Th\_SM_a, \\ Min_{i,j} &= 4_{th}inW_{i,j} - Th\_SM_a. \end{aligned} \quad (14)$$

$Max_{i,j}$ and $Min_{i,j}$ are used to determine the status of pixel $p_{i,j}$. However, in order to make the decision more precisely, we do some modifications as

$$N_{\max} = \begin{cases} Max_{i,j}, & \text{if } (Max_{i,j} \leq MedianInW_{i,j} \\ & \quad + Th\_SM_b) \\ MedianInW_{i,j} & \\ \quad + Th\_SM_b, & \text{otherwise.} \end{cases} \quad (15)$$

$$N_{\min} = \begin{cases} Min_{i,j}, & \text{if } (Min_{i,j} \geq MedianInW_{i,j} \\ & \quad - Th\_SM_b) \\ MedianInW_{i,j} & \\ \quad - Th\_SM_b, & \text{otherwise.} \end{cases} \quad (16)$$

Finally, if $f_{i,j}$ is not between N_{max} and N_{min}, we conclude that $p_{i,j}$ is a noise pixel. Edge-preserving image filter will be used to build the reconstructed value. Otherwise, the original value $f_{i,j}$ will be the output. The equation is as

$$Decision\ IV = \begin{cases} true, & \text{if } (f_{i,j} \geq N_{\max}) \text{ or } (f_{i,j} \leq N_{\min}) \\ false, & \text{otherwise.} \end{cases} \quad (17)$$

Obviously, the threshold affects the quality of denoised images of the proposed method. A more appropriate threshold contributes to achieve a better detection result. However, it is not easy to derive an optimal threshold through analytic formulation. The fixed values of thresholds make our algorithm simple and suitable for hardware implementation. According to our extensive experimental results, the thresholds Th_IM_a, TH_IM_b, Th_FM_a, Th_FM_b, Th_SM_a, and Th_SM_b are all predefined values and set as 20, 25, 40, 80, 15, and 60, respectively.

2.2 Edge-Preserving Image Filter

To locate the edge existing in the current W, a simple edge-preserving technique which can be realized easily with VLSI circuit is adopted. The dataflow and the pseudocode of our edge-preserving image filter are shown in Figs. 8 and 9, respectively. Here, we consider eight directional differences, from D_1 to D_8, to reconstruct the noisy pixel value, as shown in Fig. 10 and (18). Only those composed of noise-free pixels are taken into account to avoid possible misdetection. Directions passing through the suspected pixels are discarded to reduce misdetection. Therefore, we use $Max_{i,j}$ and $Min_{i,j}$, defined in similarity module, to determine whether the values of d, e, f, g, and h are likely corrupted, respectively. If the pixel is likely being corrupted by noise, we don't consider the direction including the suspected pixel. In the second block, if d, e, f, g, and h are all suspected to be noisy pixels, and no edge can be processed, so $\hat{f}_{i,j}$ (the

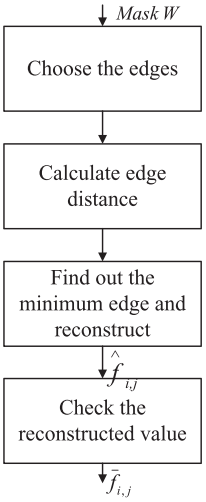


Fig. 8. Dataflow of edge-preserving image filter.

estimated value of $p_{i,j}$) is equal to the weighted average of luminance values of three previously denoised pixels and calculated as $(a + b \times 2 + c)/4$. In other conditions, the edge filter calculates the directional differences of the chosen directions and locates the smallest one (D_{min}) among them in the third block. The equations are as follows:

$$\begin{aligned} D_1 &= |d-h| + |a-e| \\ D_2 &= |a-g| + |b-h| \\ D_3 &= |b-g| \times 2 \\ D_4 &= |b-f| + |c-g| \\ D_5 &= |c-d| + |e-f| \\ D_6 &= |d-e| \times 2 \\ D_7 &= |a-h| \times 2 \\ D_8 &= |c-f| \times 2, \end{aligned} \tag{18}$$

$$\hat{f}_{i,j} = \begin{cases} (a+d+e+h)/4, & \text{if } D_{\min} = D_1, \\ (a+b+g+h)/4, & \text{if } D_{\min} = D_2, \\ (b+g)/2, & \text{if } D_{\min} = D_3, \\ (b+c+f+g)/4, & \text{if } D_{\min} = D_4, \\ (c+d+e+f)/4, & \text{if } D_{\min} = D_5, \\ (d+e)/2, & \text{if } D_{\min} = D_6, \\ (a+h)/2, & \text{if } D_{\min} = D_7, \\ (c+f)/2, & \text{if } D_{\min} = D_8. \end{cases} \tag{19}$$

In the last block of Fig. 8, the smallest directional difference implies that it has the strongest spatial relation with $p_{i,j}$, and probably there exists an edge in its direction. Hence, the mean of luminance values of the pixels which possess the smallest directional difference is treated as $\hat{f}_{i,j}$. After $\hat{f}_{i,j}$ is determined, a tuning skill is used to filter the bias edge. If $\hat{f}_{i,j}$ obtain the correct edge, it will situate at the median of b, d, e, and g because of the spatial relation and the characteristic of edge preserving. Otherwise, the values of $\bar{f}_{i,j}$ will be replaced by the median of four neighboring pixels (b, d, e, and g). We can express $\bar{f}_{i,j}$ as

$$\bar{f}_{i,j} = \text{Median}(\hat{f}_{i,j}, b, d, e, g). \tag{20}$$

```
/ * Edge - Preserving Image Filter * /
  if (d, e, f, g and h are all suspected to be noisy pixels, )
      f̂_{i,j} = (a + 2×b + c) / 4;      / * no edge is considered * /
  else
     { Find D_min (the smallest directional difference among the chosen directions);
       f̂_{i,j} = the mean of luminance values of the pixels which own D_min ; }
 / * To preserve the edge precisely, a concept like standard median filter is used. * /
  f̄_{i,j} = Median(f̂_{i,j}, b, d, e, g).
```

Fig. 9. Pseudocode of edge-preserving image filter.

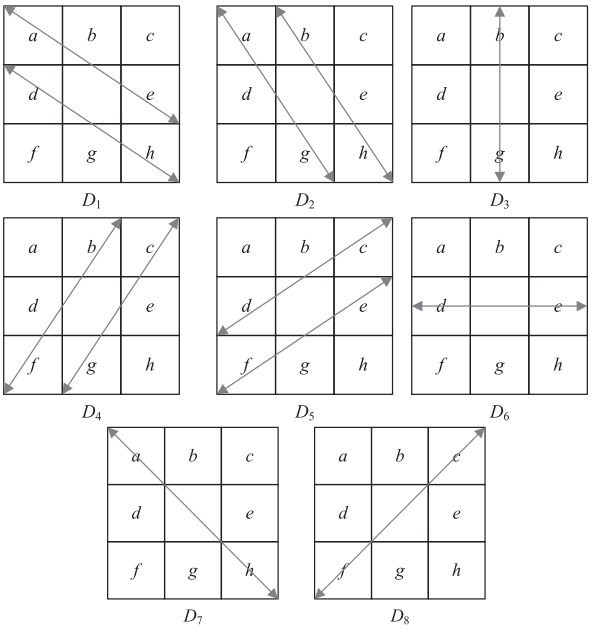


Fig. 10. Eight directional differences of DTBDM.

3 VLSI Implementation of DTBDM

DTBDM has low computational complexity and requires only two line buffers instead of full images, so its cost of VLSI implementation is low. For better timing performance, we adopt the pipelined architecture to produce an output at every clock cycle. In our implementation, the SRAM used to store the image luminance values is generated with the $0.18\ \mu$m TSMC/Artisan memory compiler [31], and each of them is 512×8 bits. According to the simulation results obtained from DesignWare of SYNOPSYS [32], we find that the access time for SRAM is about 5 ns. Hence, we adopt the 7-stage pipelined architecture for DTBDM.

Fig. 11 shows block diagram of the VLSI architecture for DTBDM. The architecture adopts an adaptive technology and consists of five main blocks: line buffer, register bank (RB), decision-tree-based impulse detector, edge-preserving image filter, and controller. Each of them is described briefly in the following sections.

3.1 Adaptive Technology

The proposed method employs an adaptive technology to improve the quality of reconstructed image. Fig. 11 shows the architecture of the proposed adaptive algorithm. The reconstructed pixels are adaptively written or stored into the line buffers. The current pixel to be denoised is located at coordinate (i, j). The adaptive points located at coordinate $(i-1, j-1)$, $(i-1, j)$, and $(i-1, j+1)$ are already denoised at the previous denoising process. Since we adopt a pipelined

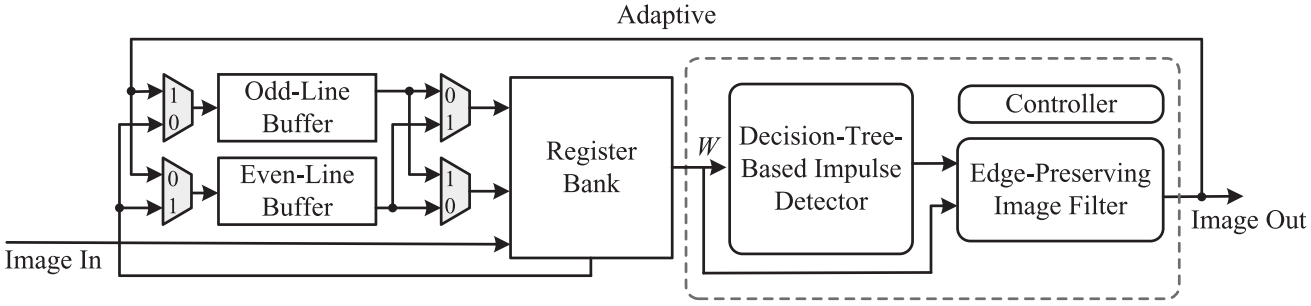


Fig. 11. Block diagram of VLSI architecture of DTBDM.

hardware architecture in this work, the denoised value located at coordinate $(i, j-1)$ is still in the pipeline and not available, as shown in Fig. 12. Here, six original points and three adaptive points are combined as nine input pixels for the 3×3 mask to reconstruct the resulting pixel $\bar{f}_{i,j}$. Although only three adaptive points (less than half) of nine input points are available, the reconstructed value is significantly affected by the adaptive points, since each adaptive point is written back to influence the next point continuously.

3.2 Line Buffer (Ping-Pong Buffer)

DTBDM adopts a 3×3 mask, so three scanning lines are needed. If $p_{i,j}$ are processed, three pixels from row_{i-1}, row_i, and row_{i+1}, are needed to perform the denoising process. Here, we use the concept of ping-pong arrangement. With the help of four crossover multiplexers (see Fig. 12), we realize three scanning lines with two line buffers. Odd-Line Buffer and Even-Line Buffer are designed to store the pixels at odd and even rows, respectively, as shown in Fig. 12.

To reduce cost and power consumption, the line buffer is implemented with a dual-port SRAM (one port for reading out data and the other for writing back data concurrently) instead of a series of shifter registers. If the size of an image is $I_w \times I_h$, the size required for one line buffer is $I_w - 3$ bytes in which 3 represents the number of pixels stored in the register bank.

3.3 Register Bank

The register bank, consisting of nine registers, is used to store the 3×3 pixel values of the current mask W. Fig. 12 shows its architecture where each three registers are connected serially in a chain to provide three pixel values of a row in W, and the Reg4 keeps the luminance value $(f_{i,j})$ of the current pixel to be denoised. Obviously, the denoising process for $p_{i,j}$ doesn't start until $f_{i+1,j+1}$ enterers from the input device. The nine values stored in RB are then used simultaneously by subsequent data detector and noise filter for denoising.

Once the denoising process for $p_{i,j}$ is completed, the reconstructed pixel value $\bar{f}_{i,j}$ generated by the edge-preserving filter is outputted and written into the line buffer storing row_i to replace $f_{i,j}$. When the denoising process shifts from $p_{i,j}$ to $p_{i,j+1}$, only three new values $(f_{i-1,j+2}, f_{i,j+2}, f_{i+1,j+2})$ are needed to be read into RB (Reg2, Reg5, and Reg8, respectively) and other six pixel values are shifted to each one's proper register. At the same time, the previous input value from the input device, $f_{i+1,j+1}$, is written back to the line buffer storing row_{i-1} for subsequent denoising process.

The selection signals of the four multiplexers are all set to 1 or 0 for denoising the odd or the even rows, respectively. Two examples are shown in Fig. 13 to illustrate the interconnections between the two line buffers and RB. Assume that we denoise row_2, and set all four selection signals to 0, those samples of row_1 and row_2 are stored in Odd-Line Buffer and Even-Line Buffer, respectively. The

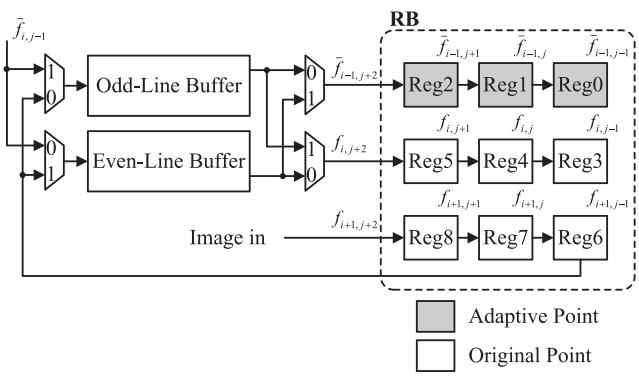


Fig. 12. Architecture of register bank in DTBDM.

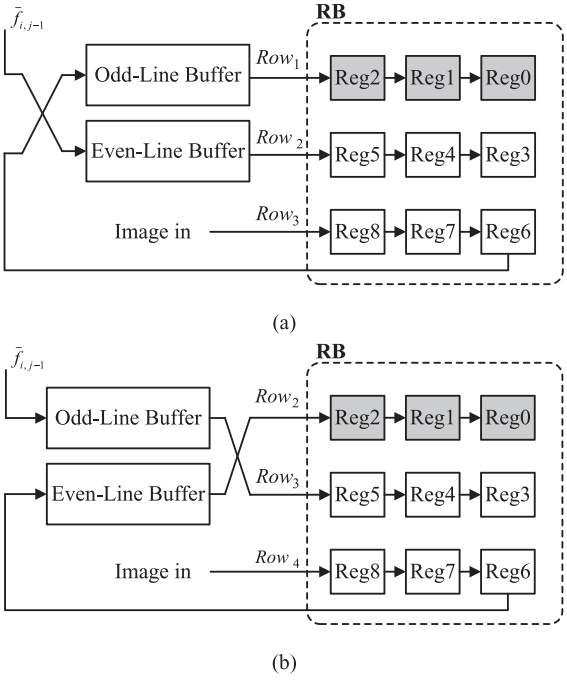


Fig. 13. Two examples of the interconnections between two line buffers and RB.

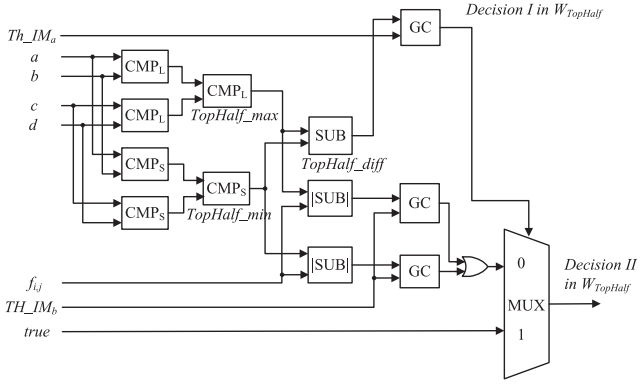


Fig. 14. Architecture of IM $(W_{TopHalf})$.

samples of row_3 are inputted from the input device, as shown in Fig. 13a. The previous value in Reg6 and the denoised results generated by the edge-preserving filter are written back to Odd-Line Buffer and Even-Line Buffer, respectively. After the denoising process of row_2 has been completed, Odd-Line Buffer is now full with the whole samples of row_3, while those denoised samples of row_2 are all stored in Even-Line Buffer. To denoise row_3, we set all selections signals to 1. Thus, the previous value in Reg6 and the denoised results generated by the edge-preserving filter are written back to Even-Line Buffer and Odd-Line Buffer, respectively, as shown in Fig. 13b. After the denoising process of row_3 has been completed, Even-Line Buffer is now full with the whole samples of row_4, while those denoised samples of row_3 are stored in Odd-Line Buffer.

3.4 Decision-Tree-Based Impulse Detector

The decision-tree-based impulse detector is composed of three modules (isolation module, fringe module, and similarity module). Each of them is described in the following sections.

3.4.1 Isolation Module

Fig. 14 shows the architecture of IM (we take $W_{TopHalf}$ for example). The comparator $\mathrm{CMP_L}$ is used to output the larger value from the two input values while the comparator $\mathrm{CMP_S}$ is used to output the smaller value from the two input values. The first two-level comparators are used to find *TopHalf_max* and *TopHalf_min*. The SUB unit is used to output the difference which is subtracted the lower input (*TopHalf_min*) from the upper one (*TopHalf_max*), and the |SUB| unit is used to output the absolute value of difference of two inputs. The GC is the greater comparator that will output logic 1 if the upper input value is greater than the lower one. The OR gate is employed to generate the binary result for *IM_TopHalf*. Finally, if the result of *Decision II* is positive, $p_{i,j}$ might be a noisy pixel or situate on an edge. The next module (FM) will be used to confirm the result.

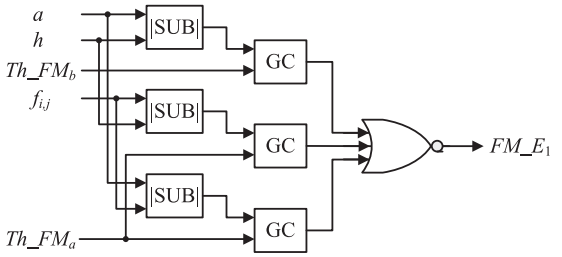


Fig. 16. Architecture of $\mathrm{FM\_1}$ module.

3.4.2 Fringe Module

Fig. 15 shows the architecture of FM. The FM is composed of four small modules, from FM_1 to FM_4, and each of them is used to determine its direction, as mentioned in Section 2.1.2. Fig. 16 is a detailed implementation of FM_1. Since E_1 is the direction from a to h (Fig. 7), the relation between a, h, and $f_{i,j}$ must be referenced. The three |SUB| units are used to determine the absolute differences between them. The GC is described in the above section and the NOR gate is used to generate the result of FM_E_1. If the result is positive, we consider that $f_{i,j}$ is on the edge E_1 and regard it as noise free.

3.4.3 Similarity Module

If IM and FM can't determine whether $f_{i,j}$ belongs to a noise-free value or not, SM is used to confirm the result. Fig. 17 shows our architecture that is designed to accelerate the sorting speed to obtain the fourth value in mask W. The detailed implementation of module M0 is shown in Fig. 18. If a is greater than b, $C01$ is set to 1; otherwise, $C01$ is set to 0. The eight GC units are used to determine the values from $C01$ to $C08$. After comparing, a combined unit is used to combine the results of each comparator to obtain a number between 0 and 8. The number indicates the order of value in mask W. If a is the smallest value in mask W, the output of the M0 module is 0; if a is the biggest value in mask W, the output is 8. The architectures of other modules (M1 to M8) are almost the same as M0, with only little difference. By means of this implementation, we can find out the order of

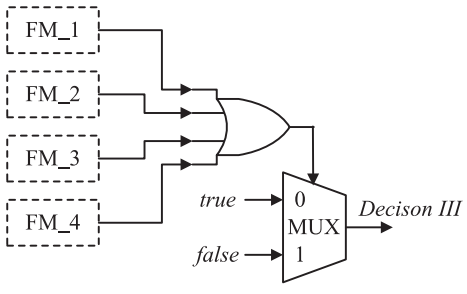


Fig. 15. Architecture of FM.

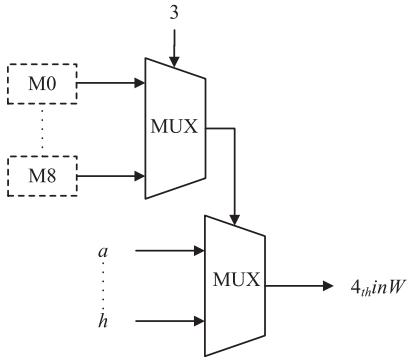


Fig. 17. Architecture of sorting $(4_{\mathrm{th}}\mathrm{inW})$.

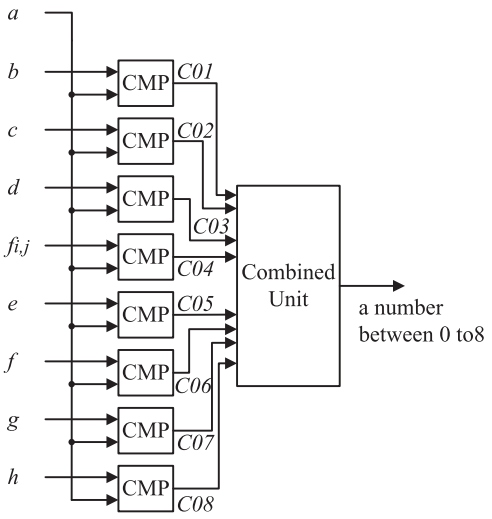


Fig. 18. Architecture of M0 module.

values efficiently by using simple comparators and combined units. Comparing with traditional sorting algorithms, this method not only speeds up the sorting time, but also reduces the space which used to store the value to be exchanged. Fig. 19 shows the architecture of SM after we obtain the fourth, fifth, and sixth values in mask W. The SUB and ADD units are used to calculate the value of $Max_{i,j}$ and $Min_{i,j}$ as mentioned in Section 2.1.3. The Two GC and MUX units are used to determine the N_{max} and N_{min}. The TC unit is a triple input comparator which can output the logic 1 if the lowest input is not between the upper two inputs.

3.5 Edge-Preserving Image Filter

The Edge-Preserving Image Filter is composed of two modules, minED generator and average generator (AG). Fig. 20 shows the architecture of the minED generator which is used to determine the edge that has the smallest difference. Eight directional differences are calculated with $twelve$ |SUB|, four ADD, and four shifter units. Then, the smallest one is determined by using the Min Tree unit. Min Tree is made up of a series of comparators. After that, the mean of luminance values of the pixels which process the smallest directional difference (D_{min}) can be obtained from the average generator, as shown in Fig. 21. As mentioned in Section 2, if $p_{i,j-1}, p_{i,j+1}, p_{i+1,j-1}, p_{i+1,j}$ and $p_{i+1,j+1}$ are all suspected to be noisy pixels, the final MUX will output $(\mathrm{a} + \mathrm{b} \times 2 + \mathrm{c})/4$. Otherwise, the MUX will output the mean of the pixel values which process D_{min}. Some directional differences are determined according to four pixel values, so its reconstructive values also need four pixel values.

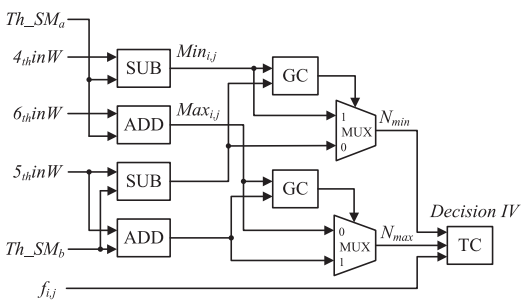


Fig. 19. Architecture of SM.

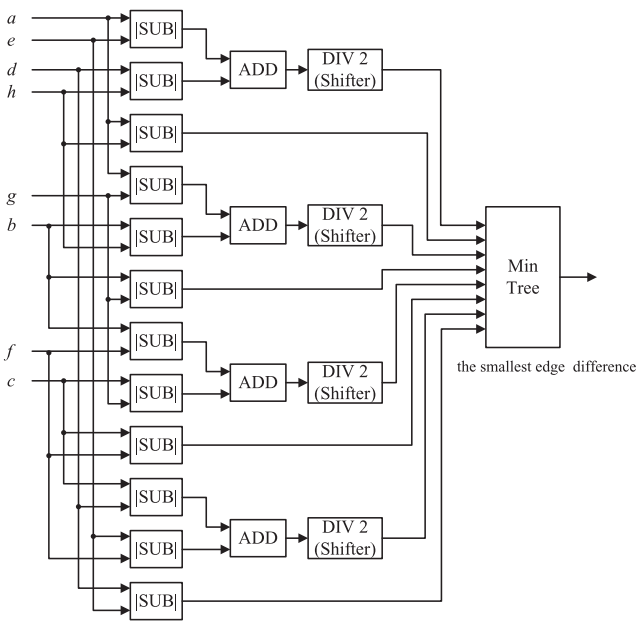


Fig. 20. Architecture of minED generator.

Two-level ADD and shifter units are used to complete our calculation. As for the chip implementation, the gate counts or silicon area of one multiplier or division is much larger than one shifter. In our design, all multipliers or divisions will be replaced by shifter units in order to lower the hardware cost.

After above computations, we sort b, d, e, and g in order. The reconstructed value $\hat{f}_{i,j}$ obtained from edge-preserving filter will be compared with the second and third values, named as *SortFour*2, *SortFour*3, and the final value $\bar{f}_{i,j}$ is obtained from the equation as

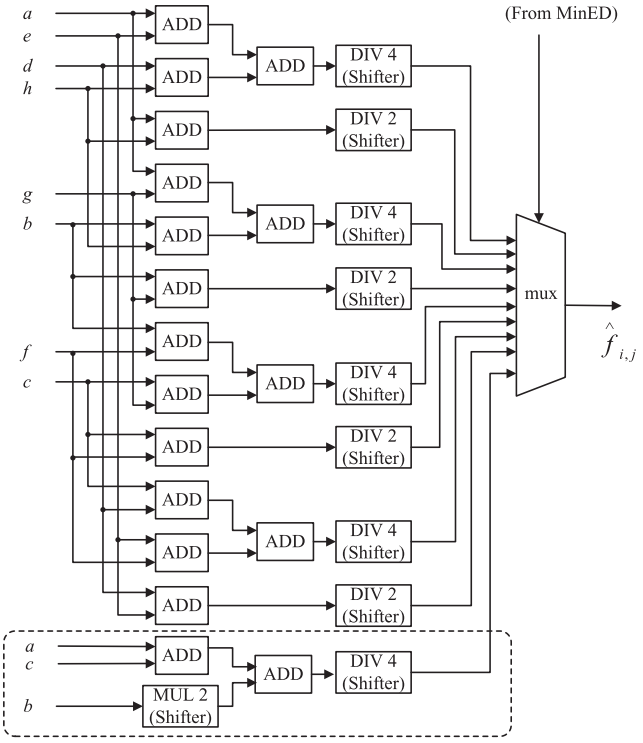


Fig. 21. Architecture of average generator.

TABLE 1
Comparative Results in PSNR (dB) of Images Corrupted by 5 Percent Impulses

	images / method	Comparative Results					
		Lena	Boat	Couple	Goldhill	Peppers	Airplane
L.C	Noisy	22.25	22.03	22.45	22.20	22.01	21.12
	median [8]	33.19	30.25	29.11	30.74	33.81	32.66
	ACWM [12]	39.96	35.58	33.39	35.67	39.34	38.25
	MSM (3:T) [13]	36.93	35.16	33.93	35.28	36.88	36.24
	MSM (5:T) [13]	36.82	35.31	34.25	35.38	36.78	36.26
	MSM (7:T) [13]	35.79	34.83	34.34	34.92	35.94	35.26
	RVNP [33]	36.29	32.17	30.75	32.45	36.23	35.02
	AMF [34]	37.50	33.19	31.88	33.66	37.30	36.26
	NAVF [35]	37.83	33.45	31.92	33.72	37.65	36.33
	LCNR [36]	38.67	34.10	32.23	34.90	39.08	37.15
	Proposed	41.15	36.83	34.63	37.26	39.84	38.63
H.C	ATMBM [14]	38.72	34.61	32.72	35.28	39.43	37.49
	DRID [15]	39.39	35.98	34.17	35.84	38.92	38.53
	RORD-WMF[16]	37.52	32.34	30.39	33.91	36.69	35.13

L.C= Lower Complexity;H.C= Higher Complexity

TABLE 2
Comparative Results in PSNR (dB) of Images Corrupted by 10 Percent Impulses

	images / method	Comparative Results					
		Lena	Boat	Couple	Goldhill	Peppers	Airplane
L.C	Noisy	19.18	19.03	19.42	19.07	18.99	18.06
	median [8]	32.53	29.76	28.55	30.30	33.04	31.78
	ACWM [12]	37.29	34.06	32.17	34.39	36.93	35.87
	MSM (3:T) [13]	33.65	32.46	31.67	32.54	33.42	32.81
	MSM (5:T) [13]	33.42	32.38	31.71	32.43	33.22	32.58
	MSM (7:T) [13]	31.42	30.69	30.66	30.70	31.22	30.41
	RVNP [33]	32.72	31.26	29.03	31.55	33.82	32.08
	AMF [34]	33.96	32.55	30.18	32.59	34.33	33.01
	NAVF [35]	34.08	32.58	30.89	32.72	34.57	33.14
	LCNR [36]	36.76	33.02	31.22	33.82	36.94	35.36
	Proposed	38.22	34.48	32.86	35.14	37.18	36.30
H.C	ATMBM [14]	36.82	33.26	31.70	33.76	36.81	35.13
	DRID [15]	36.84	34.20	32.78	34.33	36.55	36.03
	RORD-WMF[16]	36.23	31.70	29.83	33.07	35.70	34.31

L.C= Lower Complexity;H.C= Higher Complexity

$$\bar{f}_{i,j} = \begin{cases} SortFour2, & \text{if } (SortFour2 > \hat{f}_{i,j}) \\ SortFour3, & \text{if } (SortFour3 < \hat{f}_{i,j}) \\ \hat{f}_{i,j}, & \text{otherwise.} \end{cases} \quad (21)$$

3.6 Controller

Controller sends signals to control pipelining and timing statuses of the proposed circuits. It also sends control signals to schedule reading and writing statuses of the data that are stored in register bank or in line buffers. The realization of the controller is based on the concept of finite state machine (FSM). By the controller design, the proposed circuit can automatically receive stream-in data of original images and produce stream-out results of reconstructed images.

4 Implementation Results and Comparisons

To verify the characteristics and the quality of denoised images of various denoising algorithms, a variety of simulations are carried out on the six well-known 512×512 8-bit gray-scale test images: Lena, Boat, Couple, Peppers, Airplane, and Goldhill. For a single test image, the corrupted versions of it are generated in Matlab environment with random-valued impulse noise at various noise densities from 5 to 20 percent with increments of 5 percent. Then, we employ different approaches to detect impulse noise and restore the corrupted image. Thus, we can easily compare the restored images with the source image for various denoising methods.

Totally, 12 denoising methods (Median Filter [8], ACWM [12], MSM(3:T) [13], MSM(5:T) [13], MSM(7:T) [13], ATMBM [14], DRID [15], RORD-WMF [16], RVNP [33], AMF [34], NAVF [35], LCNR [36]) and our method (DTBDM) are compared in terms of objective testing (quantitative evaluation) and subjective testing (visual quality) where the parameters or thresholds of these methods are set as suggested.

We employ the peak signal-to-noise ratio (PSNR) to illustrate the quantitative quality of the reconstructed images for various methods. Tables 1, 2, 3, and 4 list the restoration results in PSNR (dB) of test images corrupted by 5, 10, 15, and 20 percent impulses, respectively. The first 10 belong to lower complexity methods and perform no iteration. The others belong to higher complexity methods and have more complicated operations and iterations. Comparing with those experimental results, the quantitative qualities of DTBDM are always better than those lower complexity methods in low noise ratio and almost the same with other higher complexity methods. Table 5 lists the mask size, line buffer, and iteration times of each methods. Since most methods are software implementation and some have complex operations, it is hard to list the detailed operations, such as the numbers of adder, subtraction, multiplication, and division. Generally, the cost of VLSI implementation depends mainly on the required memory and computational complexity. The proposed design requires only few computations. Hence, we use line buffer and iteration times to prove that our design requires lower cost. In [14], the author claims that their method achieves better filtering quality with lower complexity than other methods [9], [10], [11]. Therefore, we can conclude that our DTBDM outperforms other methods [8], [9], [10], [11], [12], [13], [14], [15], [16], [33], [34], [35], [36] in terms of quantitative evaluation.

TABLE 3
Comparative Results in PSNR (dB) of Images Corrupted by 15 Percent Impulses

	images / method	Comparative Results					
		Lena	Boat	Couple	Goldhill	Peppers	Airplane
L.C	Noisy	17.46	17.27	17.67	17.30	17.16	16.32
	median [8]	31.81	29.07	28.05	29.85	32.13	31.78
	ACWM [12]	35.15	32.37	30.80	32.92	34.57	33.63
	MSM (3:T) [13]	31.43	30.39	29.71	30.46	31.15	30.52
	MSM (5:T) [13]	30.97	30.03	29.53	30.14	30.60	29.97
	MSM (7:T) [13]	28.33	27.60	27.76	27.79	27.79	27.00
	RVNP [33]	32.25	29.45	28.77	30.03	32.72	31.94
	AMF [34]	33.47	30.66	29.79	31.28	33.19	32.78
	NAVF [35]	33.80	30.72	29.84	31.41	33.43	32.94
	LCNR [36]	35.38	31.95	30.29	32.86	35.25	33.95
	Proposed	36.17	32.89	31.40	33.60	35.47	34.40
H.C	ATMBM [14]	34.56	31.63	30.54	32.47	33.90	33.03
	DRID [15]	34.91	32.41	31.35	33.01	34.13	33.92
	RORD-WMF[16]	35.19	31.16	29.31	32.41	34.74	33.59

L.C= Lower Complexity;H.C= Higher Complexity

TABLE 4
Comparative Results in PSNR (dB) of Images Corrupted by 20 Percent Impulses

	images / method	Comparative Results					
		Lena	Boat	Couple	Goldhill	Peppers	Airplane
L.C	Noisy	16.21	15.99	16.40	16.12	15.91	15.05
	median [8]	30.95	28.48	27.36	29.30	31.29	29.90
	ACWM [12]	32.81	30.50	29.45	31.26	32.30	31.07
	MSM (3:T) [13]	29.30	28.35	28.09	28.56	29.11	28.14
	MSM (5:T) [13]	28.66	27.82	27.70	28.02	28.41	27.43
	MSM (7:T) [13]	25.59	24.97	25.32	27.76	25.08	24.08
	RVNP [33]	31.14	28.92	27.84	29.94	31.42	30.12
	AMF [34]	31.70	29.30	28.49	30.22	31.49	30.23
	NAVF [35]	31.91	29.43	28.73	30.74	31.73	30.66
	LCNR [36]	34.07	30.95	29.39	31.81	32.96	32.38
	Proposed	34.43	31.38	30.10	32.21	33.70	32.73
H.C	ATMBM [14]	32.23	30.13	29.18	30.90	31.62	30.49
	DRID [15]	32.86	30.87	29.88	31.40	31.80	31.62
	RORD-WMF[16]	34.87	30.57	28.77	31.76	33.91	32.74

L.C= Lower Complexity;H.C= Higher Complexity

TABLE 5
The Mask Size, Line Buffer, and Iteration Times of Each Method

	Mask Size	Line Buffer	Iteration Times
Median filter [8]	3x3	2	0
ACWM [12]	3x3	2	0
MSM (3:T) [13]	3x3	2	0
MSM (5:T) [13]	5x5	4	0
MSM (7:T) [13]	7x7	6	0
RVNP [33]	3x3	2	0
AMF [34]	3x3	4	0
NAVF [35]	3x3	4	0
LCNR [36]	3x3	2	0
Proposed	3x3	2	0
ATMBM [14]	3x3	whole image	4
DRID [15]	5x5	whole image	4
DORD-WMF [16]	7x7	whole image	3

TABLE 6
Comparisons of the Probabilities of Impulse Detection for Various Noise "Peppers" Image

	0%		5%		10%	
	miss	false-hit	miss	false-hit	miss	false-hit
ATMBM [14]	0	3582	1461	3699	3045	3792
DRID [15]	0	5151	1430	5221	2964	5273
RORD-WMF [16]	0	3794	12487	3505	24998	3230
Proposed Method	0	503	2093	514	4239	520

Table 6 shows the probabilities of impulse detection, including the false alarms and misses. The false alarm means that there is no impulse but detector says there is one. Since the quantitative qualities of our method are much better than those lower complexity methods, we only list the comparisons between our method and other higher complexity methods. Although ATMBM and DRID can reduce the numbers of misses by doing iteration, they also increase the numbers of false alarm, as shown in Table 6. The total numbers (misses + false alarms) in this work are much lower than those higher complexity methods.

To explore the visual quality, we show the reconstructed images of different denoising methods in restoring 20 percent corrupted image "Lena" and "Peppers" in Figs. 22 and 23. Since the quantitative quality of [12], [14], [15], [16], [36] and our method are better than others, we take them for visual comparisons only.

The images restored by ACWM, ATMBM, and DRID have some obvious noise. The reconstructed image of our method can preserve more details in edges (see stalk of pepper) as shown in Fig. 22. Clearly, the proposed DTBDM produces visually pleasing images.

The VLSI architecture of our design was implemented by using Verilog HDL. We used SYNOPSYS Design Vision

Fig. 22. Results of different methods in restoring 20 percent corrupted image "Lena."

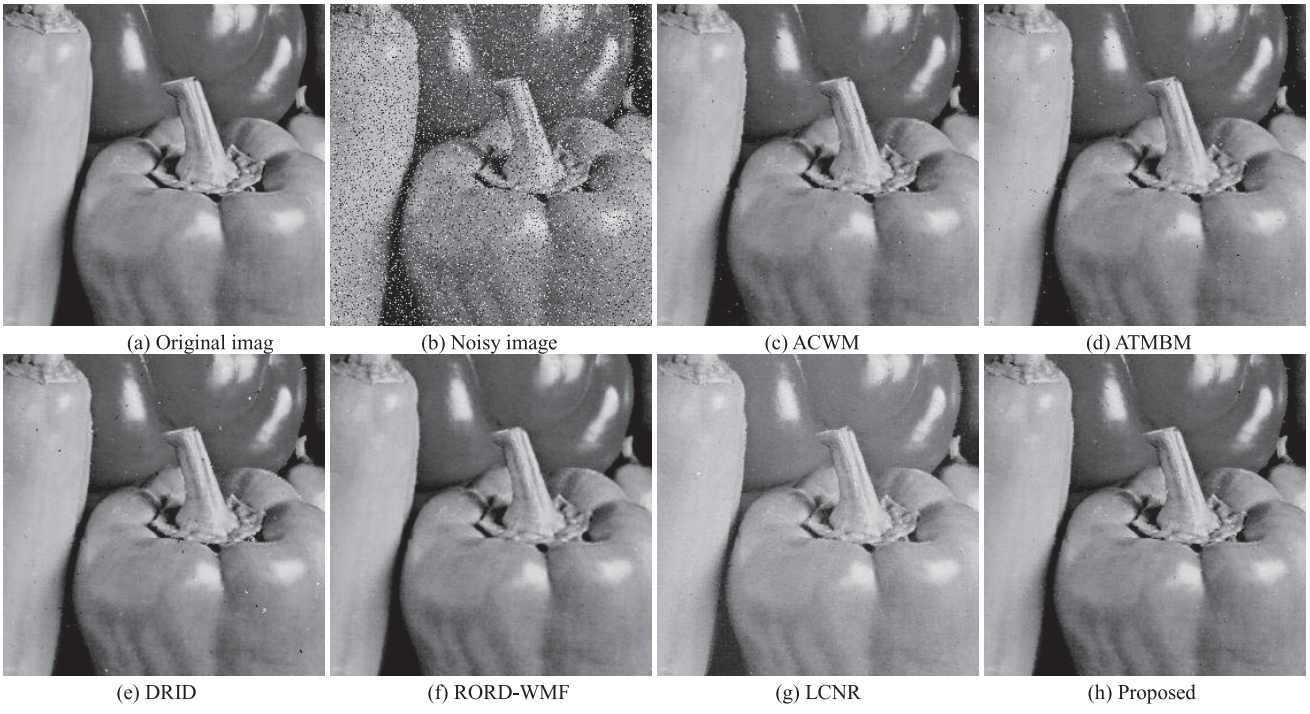


Fig. 23. Results of different methods in restoring 20 percent corrupted image "Peppers."

to synthesize the design with TSMC's 0.18 μm cell library. The layout for the design was generated with SYNOPSYS Astro (for auto placement and routing), and verified by MENTOR GRAPHIC Calibre (for DRC and LVS checks). The synthesis results show that the DTBDM chip contains 21k gate counts. It works with a clock period of 5 ns and operates at a clock rate of 200 MHz. The power consumption is 17.12 mW with 1.8-V supply voltage. Furthermore, DTBDM is also implemented on the Altera Stratix II EP2S60F1020C5 FPGA board for verification. The operating clock frequency of DTBDM is 163.83 MHz (with 1.29k logic elements). Since our design requires only simple operations (fixed bit-width adding, subtracting, or comparing), the denoised results of DTBDM with software program and with the VLSI circuit are identical.

To evaluate the circuit, we compared DTBDM with four recent denoising chips [33], [34], [35], [36]. They were realized with different VLSI implementations, so it is very difficult to compare our chip with them directly. Hence, we have implemented our design by using four different technologies, respectively. Table 7 shows the detailed comparisons for various implementations in terms of the total number of logic elements, line buffer, and frequency.

TABLE 7
Features of Five Denoising Implementations

	RVNP [33]	AMF [34]	NAVF [35]	LCNR [36]	Ours (DTBMD)			
FPGA device /ASCI Library	TSMC's 0.35 µm process	Altera FLEX10KE EPF10K200 SRC240-1	Altera STRATIX EP1S25	Altera Cyclone II EP2C20F 484C7	TSMC's 0.18 µm process	Altera FLEX10KE EPF10K200 SRC240-1	Altera STRATIX EP1S25	Altera Cyclone II EP2C20F48 4C7
Line Buffer	2	4	4	2	2	2	2	2
Area	3887 Gates	9972 logic cells	2670 logic cells	513 logic cells	21 k Gates	2166 logic cells	1743 logic cells	1709 logic cells
Frequency	45 MHz	65 MHz	97.4 MHz	129.59 MHz	200 MHz	67.14 MHz	144.11 MHz	140.37 MHz

As demonstrated, DTBDM requires lower hardware cost and works faster than RVNP [33], AMF [34], and NAVF [35]. Although the proposed method requires more logic elements than [36], it can achieve higher frequency and produce better image quality when denoising an image corrupted by random-valued impulse noise.

5 Conclusions

A low-cost VLSI architecture for efficient removal of random-valued impulse noise is proposed in this paper. The approach uses the decision-tree-based detector to detect the noisy pixel and employs an effective design to locate the edge. With adaptive skill, the quality of the reconstructed images is notable improved. Our extensive experimental results demonstrate that the performance of our proposed technique is better than the previous lower complexity methods and is comparable to the higher complexity methods in terms of both quantitative evaluation and visual quality. The VLSI architecture of our design yields a processing rate of about 200 MHz by using TSMC 0.18 μm technology. It requires only low computational complexity and two line memory buffers. Therefore, it is very suitable to be applied to many real-time applications.

Acknowledgments

This work was supported in part by the National Science Council, R.O.C., under Grant NSC-101-2221-E-006-151-MY3 and the Ministry of Economic Affairs (MOEA) of Taiwan, under Grant number MOEA 100-EC-17-A-05-S1-192. This research received funding from the Headquarters of University Advancement at the National Cheng Kung University, which is sponsored by the Ministry of Education, Taiwan, R.O.C.

REFERENCES

[1] R.C. Gonzalez and R.E. Woods, *Digital Image Processing.* Pearson Education, 2007.

[2] W.K. Pratt, *Digital Image Processing.* Wiley-Interscience, 1991.

[3] H. Hwang and R.A. Haddad, "Adaptive Median Filters: New Algorithms and Results," *IEEE Trans. Image Processing,* vol. 4, no. 4, pp. 499-502, Apr. 1995.

[4] S. Zhang and M.A. Karim, "A New Impulse Detector for Switching Median Filter," *IEEE Signal Processing Letters,* vol. 9, no. 11, pp. 360-363, Nov. 2002.

[5] R.H. Chan, C.W. Ho, and M. Nikolova, "Salt-and-Pepper Noise Removal by Median-Type Noise Detectors and Detail-Preserving Regularization," *IEEE Trans. Image Processing,* vol. 14, no. 10, pp. 1479-1485, Oct. 2005.

[6] P.E. Ng and K.K. Ma, "A Switching Median Filter with Boundary Discriminative Noise Detection for Extremely Corrupted Images," *IEEE Trans. Image Processing,* vol. 15, no. 6, pp. 1506-1516, June 2006.

[7] P.-Y. Chen and C.-Y. Lien, "An Efficient Edge-Preserving Algorithm for Removal of Salt-and-Pepper Noise," *IEEE Signal Processing Letters,* vol. 15, pp. 833-836, Dec. 2008.

[8] T. Nodes and N. Gallagher, "Median Filters: Some Modifications and Their Properties," *IEEE Trans. Acoustics, Speech, Signal Processing,* vol. ASSP-30, no. 5, pp. 739-746, Oct. 1982.

[9] S.-J. Ko and Y.-H. Lee, "Center Weighted Median Filters and Their Applications to Image Enhancement," *IEEE Trans. Circuits Systems,* vol. 38, no. 9, pp. 984-993, Sept. 1991.

[10] T. Sun and Y. Neuvo, "Detail-Preserving Median Based Filters in Image Processing," *Pattern Recognition Letters,* vol. 15, pp. 341-347, Apr. 1994.

[11] E. Abreu, M. Lightstone, S.K. Mitra, and K. Arakawa, "A New Efficient Approach for the Removal of Impulse Noise from Highly Corrupted Images," *IEEE Trans. Image Processing,* vol. 5, no. 6, pp. 1012-1025, June 1996.

[12] T. Chen and H.R. Wu, "Adaptive Impulse Detection Using Center-Weighted Median Filters," *IEEE Signal Processing Letters,* vol. 8, no. 1, pp. 1-3, Jan. 2001.

[13] T. Chen and H.R. Wu, "Space Variant Median Filters for the Restoration of Impulse Noise Corrupted Images," *IEEE Trans. Circuits Systems II, Analog Digital Signal Processing,* vol. 48, no. 8, pp. 784-789, Aug. 2001.

[14] W. Luo, "An Efficient Detail-Preserving Approach for Removing Impulse Noise in Images," *IEEE Signal Processing Letters,* vol. 13, no. 7, pp. 413-416, July 2006.

[15] I. Aizenberg and C. Butakoff, "Effective Impulse Detector Based on Rank-Order Criteria," *IEEE Signal Processing Letters,* vol. 11, no. 3, pp. 363-366, Mar. 2004.

[16] H. Yu, L. Zhao, and H. Wang, "An Efficient Procedure for Removing Random-Valued Impulse Noise in Images," *IEEE Signal Processing Letters,* vol. 15, pp. 922-925, 2008.

[17] Y. Dong and S. Xu, "A New Directional Weighted Median Filter for Removal of Random-Valued Impulse Noise," *IEEE Signal Processing Letters,* vol. 14, no. 3, pp. 193-196, Mar. 2007.

[18] N.I. Petrovic and V. Crnojevic, "Universal Impulse Noise Filter Based on Genetic Programming," *IEEE Trans. Image Processing,* vol. 17, no. 7, pp. 1109-1120, July 2008.

[19] B. De Ville, *Decision Trees for Business Intelligence and Data Mining.* SAS Publishing, 2007.

[20] S. Rasoul Safavian and D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Trans. Systems Man, Cybernetics,* vol. 21, no. 3, pp 660-674, May 1991.

[21] H.-H. Tsai, X.-P. Lin, and B.-M. Chang, "An Image Filter with a Hybrid Impulse Detector Based on Decision Tree and Particle Swarm Optimization," *Proc. IEEE Int'l Conf. Machine Learning and Cybernetics,* July 2009.

[22] H.-L. Eng and K.-K. Ma, "Noise Adaptive Soft-Switching Median Filter," *IEEE Trans. Image Processing,* vol. 10, no. 2, pp. 242-251, Feb. 2001.

[23] G. Pok, J. Liu, and A.S. Nair, "Selective Removal of Impulse Noise Based on Homogeneity Level Information," *IEEE Trans. Image Processing,* vol. 12, no. 1, pp. 85-92, Jan. 2003.

[24] Z. Wang and D. Zhang, "Progressive Switching Median Filter for the Removal of Impulse Noise from Highly Corrupted Images," *IEEE Trans. Circuits Systems II, Analog Digital Signal Processing,* vol. 46, no. 1, pp. 78-80, Jan. 1999.

[25] A.S. Awad and H. Man, "High Performance Detection Filter for Impulse Noise Removal in Images," *IEEE Electronic Letters,* vol. 44, no. 3, pp. 192-194, Jan. 2008.

[26] X. Zhang and Y. Xiong, "Impulse Noise Removal Using Directional Difference Based Noise Detector and Adaptive Weighted Mean Filter," *IEEE Signal Processing Letters,* vol. 16, no. 4, pp. 295-298, Apr. 2009.

[27] Z. Xu, H.R. Wu, B. Qiu, and X. Yu, "Geometric Features-Based Filtering for Suppression of Impulse Noise in Color Images," *IEEE Trans. Image Processing,* vol. 18, no. 8, pp. 1742-1759, Aug. 2009.

[28] F.J. Gallegos-Funes, V.I. Ponomaryov, S. Sadovnychiy, and L. Nino-de-Rivera, "Median M-Type K-Nearest Neighbour (MM-KNN) Filter to Remove Impulse Noise from Corrupted Images," *IEEE Electronics Letters,* vol. 38, no. 15, pp. 786-787, July 2002.

[29] I. Aizenberg, C. Butakoff, and D. Paliy, "Impulsive Noise Removal Using Threshold Boolean Filtering Based on the Impulse Detecting Functions," *IEEE Signal Processing Letters,* vol. 12, no. 1, pp. 63-66, Jan. 2005.

[30] Z. Wang and D. Zhang, "Restoration of Impulse Noise Corrupted Images Using Long-Range Correlation," *IEEE Signal Processing Letters,* vol. 5, no. 1, pp. 4-7, Jan. 1998.

[31] 0.18 μm TSMC/Artisan Memory Compiler, http://www.artisan.com, 2012.

[32] Synopsys Inc., "DesignWare Building Block IP," http://www.synopsys.com, 2012.

[33] S.-C. Hsia, "Parallel VLSI Design for a Real-Time Video-Impulse Noise-Reduction Processor," *IEEE Trans. Very Large Scale Integration Systems,* vol. 11, no. 4, pp. 651-658, Aug. 2003.

[34] I. Andreadis and G. Louverdis, "Real-Time Adaptive Image Impulse Noise Suppression," *IEEE Trans. Instrumentation and Measurement,* vol. 53, no. 3, pp. 798-806, June 2004.

[35] V. Fischer, R. Lukac, and K. Martin, "Cost-Effective Video Filtering Solution for Real-Time Vision Systems," *EURASIP J. Applied Signal Processing,* vol. 13, pp. 2026-2042, 2005.

[36] T. Matsubara, V.G. Moshnyaga, and K. Hashimoto, "A FPGA Implementation of Low-Complexity Noise Removal," *Proc. 17th IEEE Int'l Conf. Electronics, Circuits, and Systems (ICECS '10),* pp. 255-258, Dec. 2010.

[37] P.-Y. Chen, C.-Y. Lien, and H.-M. Chuang, "A Low-Cost VLSI Implementation for Efficient Removal of Impulse Noise," *IEEE Trans. Very Large Scale Integration Systems,* vol. 18, no. 3, pp. 473-481, Mar. 2010.

Chih-Yuan Lien received the BS and MS degrees in computer science and information engineering from National Taiwan University, Taiwan (R.O.C.), in 1996 and 1998, respectively, and the PhD degree in computer science and information engineering from National Cheng Kung University, Taiwan (R.O.C.), in 2009. He is currently an assistant professor in the Department of Electronic Engineering, National Kaohsiung University of Applied Sciences, Taiwan (R.O.C.). His research interests include image processing, VLSI chip design, and video coding system.

Chien-Chuan Huang received the BS and MS degree in computer science and information engineering from National Cheng Kung University, Taiwan (R.O.C.), in 2005 and 2008, respectively. Since September 2008, he has been working toward the PhD degree in computer science and information engineering from National Cheng Kung University, Taiwan (R.O.C.). His research interests include image processing, VLSI chip design, and data compression.

Pei-Yin Chen received the BS degree in electrical engineering from National Cheng Kung University, Taiwan (R.O.C.), in 1986, the MS degree in electrical engineering from Penn Sate University, Pennsylvania, in 1990, and the PhD degree in electrical engineering from National Cheng Kung University, Taiwan (R.O.C.), in 1999. He is currently a professor in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan (R.O.C.). His research interests include VLSI chip design, video compression, fuzzy logic control, and gray prediction. He is a member of the IEEE.

Yi-Fan Lin received the BS degree in computer science from National Sun Yat-sen University, Taiwan (R.O.C.), in 2006, and the MS degree in computer science and information engineering from National Cheng Kung University, Taiwan (R.O.C.), in 2009. His research interests include VLSI chip design, data compression, and image processing.