IET Image Processing

Research Article

IET Journals
The Institution of Engineering and Technology

Low-latency median filter core for hardware implementation of 5 × 5 median filtering

ISSN 1751-9659
Received on 6th September 2016
Revised 9th April 2017
Accepted on 30th July 2017
E-First on 14th September 2017
doi: 10.1049/iet-ipr.2016.0737
www.ietdl.org

Vineet Kumar[1] ✉, *Abhijit Asati*[1], *Anu Gupta*[1]

[1]*Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani 333031, India*

✉ *E-mail: vineet.bitsp@gmail.com*

Abstract: This study presents hardware implementation of 5 × 5 median filter that uses a new low-latency median filter (LLMF) core in order to find the median of 25 integer values. The proposed LLMF core architecture computes the median of 25 integers in just three clock cycles. The maximum frequency of operation of the proposed median filter architecture is 394 MHz on the Xilinx Zynq FPGA device. The proposed LLMF core provides reduced clock cycle latency compared with the existing state-of-the-art median filter core architectures.

1 Introduction

The median filter is a quite popular non-linear filter that is used for noise removal in digital signal and image processing applications. It is particularly effective in removing impulse or 'salt and pepper' type random noise from images while preserving the edge quality [1]. It is commonly used as a preprocessing step before applying other image processing operations such as edge detection [2].

The hardware implementation of the median filter is used in video and high-speed image acquisition cameras for adaptive denoising [3]. The median filter hardware plays an important role in the design of embedded image processing applications on field programmable logic arrays (FPGAs) [4]. In embedded applications, where low-speed central processing units (CPUs) are used, the dedicated hardware for median filtering can be very useful in order to meet real-time performance. The conventional median filtering methods, such as bubble sort, is a slow process as it involves sorting of pixels inside the filtering window that scans over the image. The commonly used sorting networks in software implementations are bubble sort, selection sort, merge sort, quick sort, and odd–even transposition sort [5–10]. The median filters of different window size, such as 3 × 3, 5 × 5, and 7 × 7 etc., are used in different applications. In [10], four different hardware implementations of 5 × 5 median filter are presented, which are based on four different sorters: Batcher sort [11], systolic sort [12], radix sort [13], and merge sort [14]. The comparative analysis of these sorters showed that the Batcher sorting network is superior to the other designs. These sorting networks when implemented in hardware result in significant latency in terms of the number of clock cycles [10]. In [3], resource-aware architecture of the median filter is presented and its latency is better than [11–14].

In this paper, a new 5 × 5 median filter core architecture is proposed in order to find the median of 25 integer values and the entire median filter is implemented on FPGA. The proposed core architecture provides low latency for the median value computation and is called 5 × 5 low-latency median filter (LLMF) core. The working of 5 × 5 median filter for processing entire image is illustrated in Fig. 1. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels. For a 5 × 5 window centred at the pixel to be processed in the input image, first all the 25 pixels are arranged in ascending (or descending) order and then median is taken that assigns to the processing pixel in the output image as shown in Fig. 1. Subsequently, this 5 × 5 window moves to the next pixel to be processed and the same median finding operation repeats. This process continues until all pixels of the input image are covered.

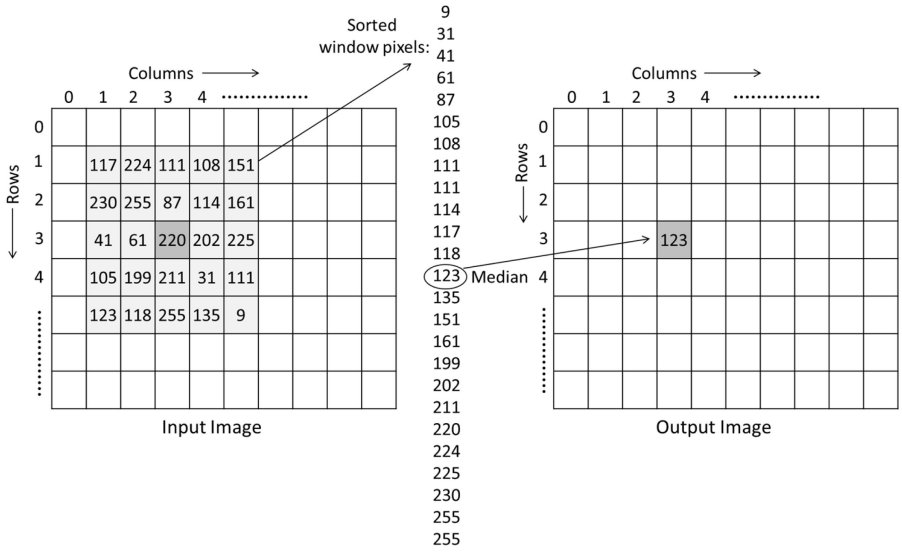


Fig. 1 *Working of 5 × 5 median filter for processing entire image*

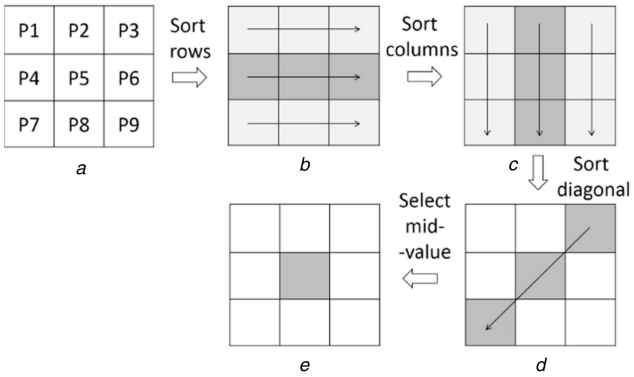


Fig. 2 *Algorithm of 3 × 3 LLMF core*
(a) Window, ***(b)*** Sorted rows, ***(c)*** Sorted columns, ***(d)*** Sorted diagonal, ***(e)*** Median of the window

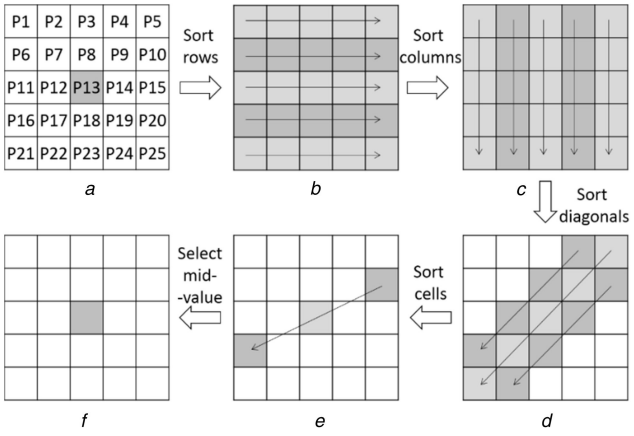


Fig. 3 *Algorithm of the proposed 5 × 5 LLMF core*
(a) Window, ***(b)*** Sorted rows, ***(c)*** Sorted columns, ***(d)*** Sorted 3 middle diagonals, ***(e)*** Sorted diagonal of (*d*), ***(f)*** Median of (*a*)

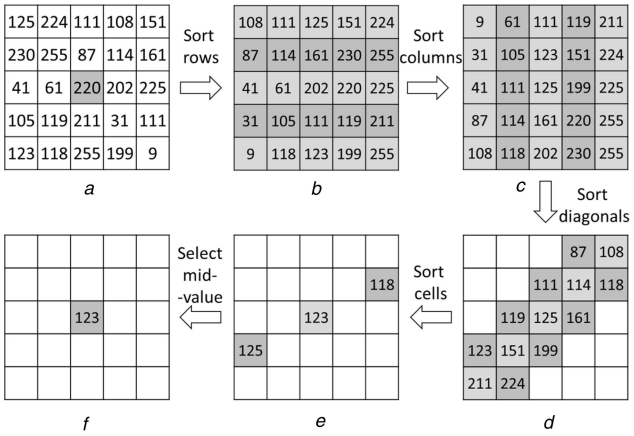


Fig. 4 *Example of median computation*
(a) A 5 × 5 window, ***(b)*** Sorted rows, ***(c)*** Sorted columns, ***(d)*** Sorted 3 middle diagonals, ***(e)*** Sorted diagonal of (*d*), ***(f)*** Median of (*a*)

In offline processing of signals by the median filter, only total processing time is important, but for online processing, latency is also important. For example, when a live incoming signal-stream is processed by selective switching median filtering (SSMF), where corrupted samples are detected by a noise detector and only corrupted samples undergo median filtering, the processing has to be done in real time. One such application is described in [15], where SSMF was used for online processing of music signal and it mentions that if the median filter is applied on whole music signal, its audio quality degrades.

The hardware implementation of 3 × 3 median filter was presented in [16, 17], but the higher window size, such as 5 × 5, provides improved smoothing of the image while preserving its edges, which could be advantageous for certain applications. For example, edge detection applied on a 5 × 5 median-filtered image would result in lesser false edges, which would further benefit the next level of processing, such as object detection in the edge-map [18].

The presented LLMF core architecture takes 8-bit integers as input; however, the speed of the architecture would remain the same for higher than 8-bit also, such as 16-bit, 24-bit, and so on, as the proposed architecture involves word-level processing rather than bit-level processing presented in [3, 19]. In word-level processing, all the bits of an integer (word) are processed in parallel. The latency of the proposed architecture is also independent of bit-length (*W*) of integers, whereas with an increase in *W*, the latency in [3, 19] will also increase, as they are *W*-step and *W/B*-step methods, respectively, where *B* is a design parameter having an integer value >1.

The implementation results in this paper show that the hardware resources increase drastically when moving from 3 × 3 to 5 × 5 window size using the proposed architecture approach due to the increase in the number and size of sorting networks. They will further increase largely for 7 × 7 and higher window sizes. Moreover, frequency and latency also degrade for higher window sizes. Therefore, this paper recommends the proposed architecture approach for 3 × 3 and 5 × 5 window sizes only.

The rest of this paper is organised as follows. Section 2 describes the related work. Section 3 explains in detail the proposed LLMF core for hardware implementation of 5 × 5 median filter. Section 4 presents the results and discussion. Section 5 concludes the work.

2 Existing 3 × 3 LLMF core

The LLMF core for hardware implementation of 3 × 3 median filter is presented in [16, 17] which uses a non-conventional algorithm for finding the median of nine values as shown in Fig. 2. In this algorithm, all the nine values are arranged in a 3 × 3 matrix without worrying about their order (position) in the matrix. Then, the algorithm finds the median of 3 × 3 values in three steps: (i) sorting of rows; (ii) sorting of columns; and (iii) sorting of a diagonal. The middle value of the sorted diagonal is the median value as shown in Fig. 2. This algorithm is suitable for parallel hardware implementation, because it involves independent sorting operations that can be carried out in parallel. For example, sorting of all rows or columns can be done in parallel. In [16], the hardware architecture of 3 × 3 LLMF core takes latency of six clock cycles in order to implement the algorithm depicted in Fig. 2.

The algorithm described in Fig. 2 does not work correctly for 5 × 5 window. The algorithm for 5 × 5 window is presented in the next section and this algorithm was implemented as dedicated hardware, called as the proposed 5 × 5 LLMF core. The approach in [16] gives a latency of six clock cycles for finding a median of 3 × 3 values, whereas the proposed architecture approach gives a latency of two and three clock cycles for 3 × 3 and 5 × 5 values, respectively. The proposed work presents a faster three-value sorter circuit than conventional sorter and also proposes a procedure for implementing higher value sorters using three-value sorters.

3 Proposed 5 × 5 LLMF core

In [16, 17], the median of 3 × 3 values is determined using three steps: (i) sorting of rows; (ii) sorting of columns; and (iii) sorting of a diagonal. When these three steps are applied for finding a median of 5 × 5 values, it gives the wrong median, which is then corrected by the modified algorithm presented in Fig. 3. In this algorithm, steps 1 and 2 are the same as previous, but in step 3, three diagonals are sorted instead of a single diagonal and in step 4, one diagonal of three values is sorted. The single diagonal in step 4 as shown in Fig. 3*e* is in fact the diagonal of the three diagonals sorted in the previous step. This diagonal consists of three values (one value from each of the three diagonals) as shown in Figs. 3 and 4.

Fig. 4 illustrates the algorithm of the proposed 5 × 5 LLMF core with help of an example. In order to find the median of 25 values, all the values are arranged as a 5 × 5 matrix without worrying about

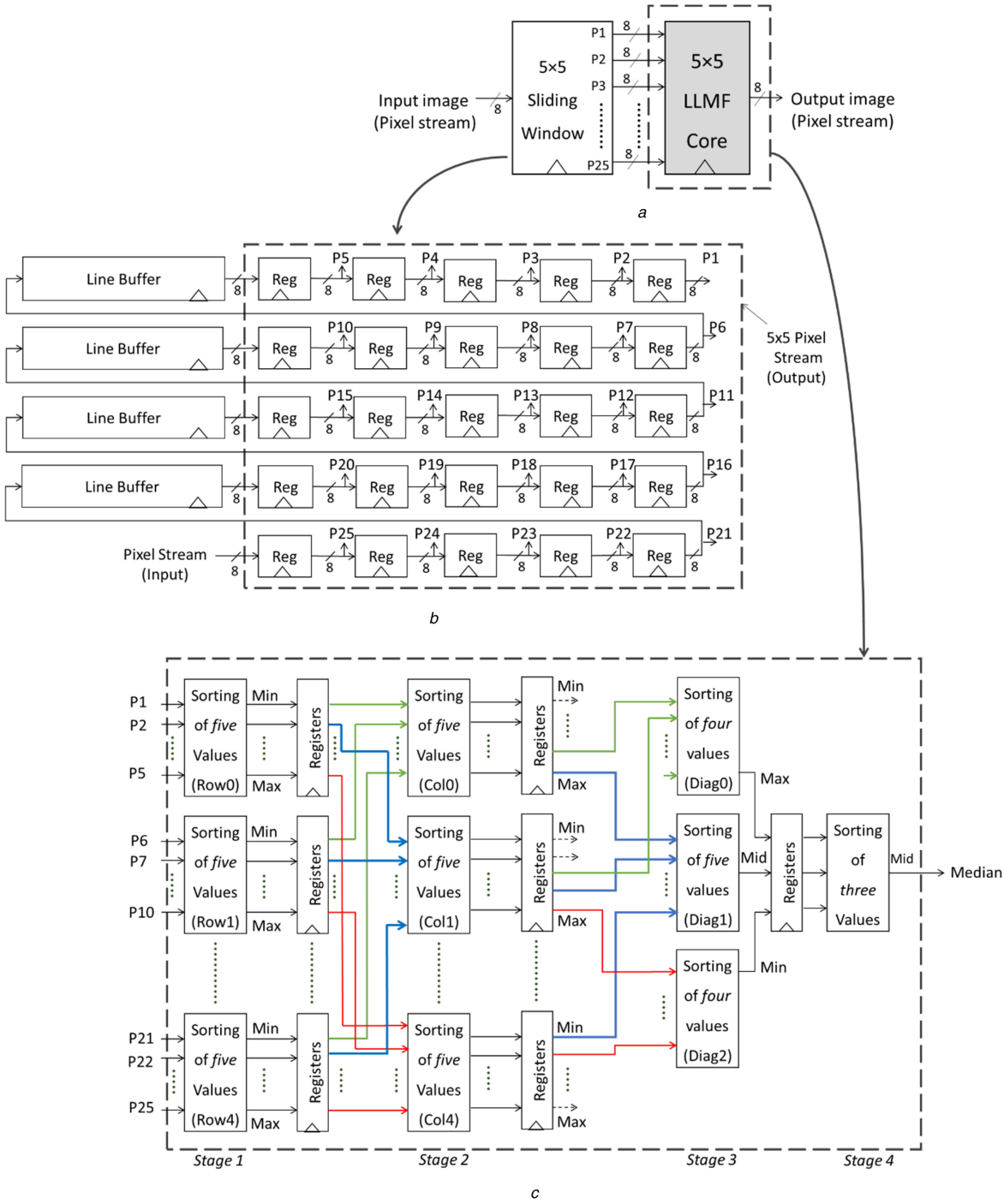


Fig. 5 *Proposed filter*
(a) 5 × 5 median filter, ***(b)*** 5 × 5 sliding window architecture, ***(c)*** Proposed 5 × 5 LLMF core

their order (position) in the matrix. The procedure shown in Fig. 4 involves four steps. In the first step, each of the five rows of the matrix is sorted and the sorted rows are placed in new matrix shown in Fig. 4*b*. In the second step, each of the five columns of the new matrix is sorted as shown in Fig. 4*c*. From this column-sorted matrix, three middle diagonals are sorted as shown in Fig. 4*d*. Now, a three-value diagonal is extracted from these three diagonals selecting one value from each diagonal as shown in Fig. 4*e*. Finally, this three-value diagonal is sorted and the middle value is the median. The white cells in Figs. 4*d* and *e* are not involved in the computations.

The 5 × 5 median filter is shown in Fig. 5*a*, which uses the proposed 5 × 5 LLMF core in order to find the median of 25 values. The input image is applied as a pixel stream (one pixel per clock cycle) to the median filter and it produces pixel stream of the median filtered image. The sliding window provides 5 × 5 pixels (P1, P2, P3,…, P25) of the input image to the LLMF core. The sliding window effect is realised using the architecture shown in Fig. 5*b*, which is called sliding window architecture. This architecture uses four line (row) buffers that are essentially four FIFO (first in, first out) register sets, and 25 8-bit registers. The length of each line buffer is equal to WIDTH-5 for an image width, WIDTH. This sliding window architecture takes a pixel stream of the image as input and provides a 5 × 5 pixel stream (5 × 5 pixels per clock cycle) as output, after an initial delay of (4WIDTH + 5) clock cycles. The output, 5 × 5 pixel stream (P1, P2, P3,…, P25), is applied as input to the LLMF core, which gives a median (output) per clock cycle, after an initial latency of 3. The architecture of the proposed 5 × 5 LLMF core shown in Fig. 5*c* is described in the following subsection.

3.1 Hardware architecture

The hardware architecture of the proposed 5 × 5 LLMF core is shown in Fig. 5*c* and its algorithm was discussed in the beginning of Section 3. This architecture is composed of four stages: (i) rows sort; (ii) columns sort; (iii) three diagonals sort; and (iv) one three-value diagonal sort. In stages 1 and 2 of the architecture, each of rows or columns is sorted using 'sorting of five values' component, whereas stage 3 requires both sorting of four and five values in order to sort two diagonals of four values each and one diagonal of five values. Finally, in stage 4, sorting of three values (three-value sorter) is used for single diagonal sort. All the rows, columns, or diagonals in a stage are sorted in parallel. In order to get throughput of one median per clock cycle, the pipelined registers are introduced in between different stages of the architecture, which gives a pipeline depth of 3. The sorted values are stored in these registers, which act as input for next stage. The sorting of three, four, and five values in Fig. 5*c* is realised using combinational logic circuits. The sorting of five or four values is carried out using a three-value sorter as explained below.

3.1.1 Sorting of five values: The proposed method for 'sorting of five values' using a three-value sorter is illustrated in Fig. 6 with help of an example. This sorting completes in four steps. For sorting n values using a three-value sorter, it will take $n-1$ steps (iterations) to complete.

The combinational circuit for 'sorting of five values' is shown in Fig. 7. The circuit takes five 8-bit values as input, which are a, b, c, d, and e, and provides these values in ascending order as v, w, x, y, and z. The inputs are applied to three-value sorters in a specific order as per the procedure described in Fig. 6. The proposed 5 × 5 LLMF core architecture also uses a 'sorting of four values' component, which is implemented using 3 three-value sorters, as sorting of four numbers completes in three steps as per procedure depicted in Fig. 6.

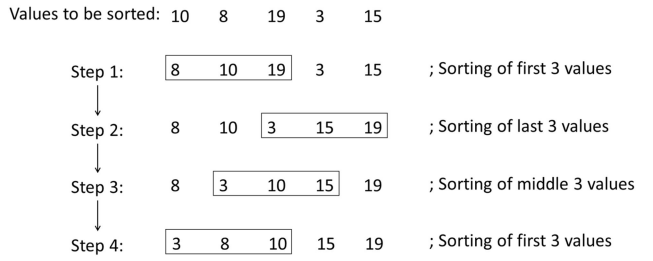


Fig. 6 *Example of the proposed procedure for 'sorting of five values' using a three-value sorter*

3.1.2 Three-value sorter: The three-value sorter is the fundamental component of the proposed 5 × 5 LLMF core, as all the higher values sorters have been implemented using it. This component decides the overall speed and resource utilisation of the proposed core architecture. The conventional three-value sorter is shown in Fig. 8, which is implemented using three two-value sorters [20, 21]. This is a three-level serial implementation, where the output of a two-value sorter in one level is used as the input for other two-value sorter in the next level.

In contrast to the conventional three-value sorter shown in Fig. 8, the proposed three-value sorter shown in Fig. 9 is a parallel implementation, which performs all the comparison operations in parallel. This combinational circuit is mainly designed using three two-value comparators and three multiplexers. The proposed three-value sorter is faster than conventional three-value sorter, as it performs all the comparison operations in parallel. This improves the overall maximum frequency of operation of the proposed 5 × 5 LLMF core, as the speeds of all the sorting modules and the overall architecture depend on the propagation delay of three-value sorter.

4 Results and discussion

In this section, FPGA implementation and performance results of the proposed median filter are presented, and the proposed LLMF core is compared with other state-of-the-art architectures.

4.1 FPGA implementation results

The proposed median filter was implemented with Verilog HDL to target Xilinx's 7 series FPGA, Zynq xc7z020-1clg484. Table 1 shows the synthesis results of the proposed architecture shown in Fig. 5. The overall maximum frequency, f_{max}, obtained is 394 MHz. Two block RAMs were used to realise four line buffers of 5 × 5 sliding window architecture for the image width, WIDTH = 640. For larger image widths, only the number of block RAMs may increase without any change in slice registers and slice LUTs listed in Table 1. The target FPGA contains 140 block RAMs and size of each block RAM is 36 K bits. Each block RAM can be configured as a dual port RAM of aspect ratio 2 K × 16-bits. Each block RAM

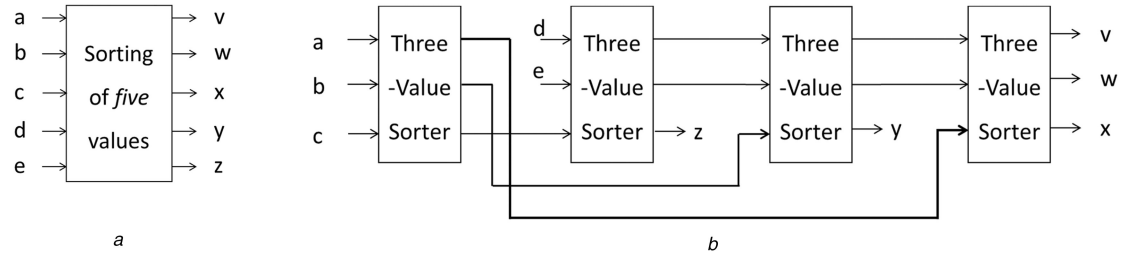


Fig. 7 *Sorting of five values*
(a) Sorting module, ***(b)*** Circuit that uses three-value sorter

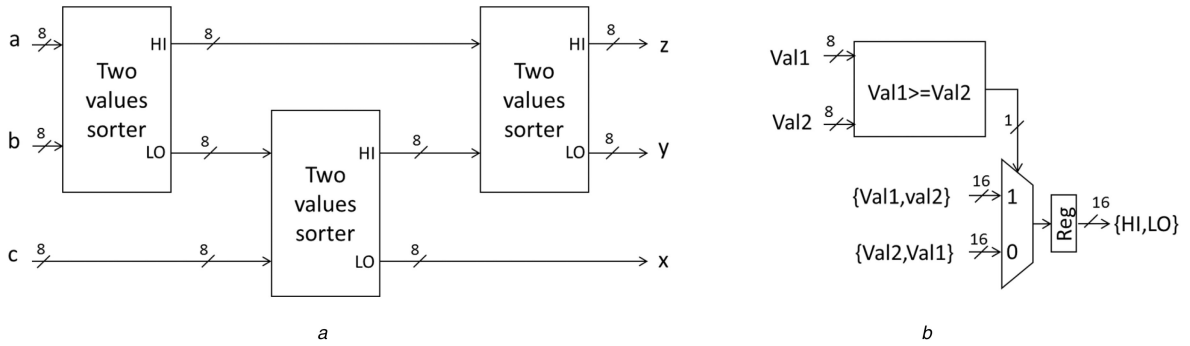
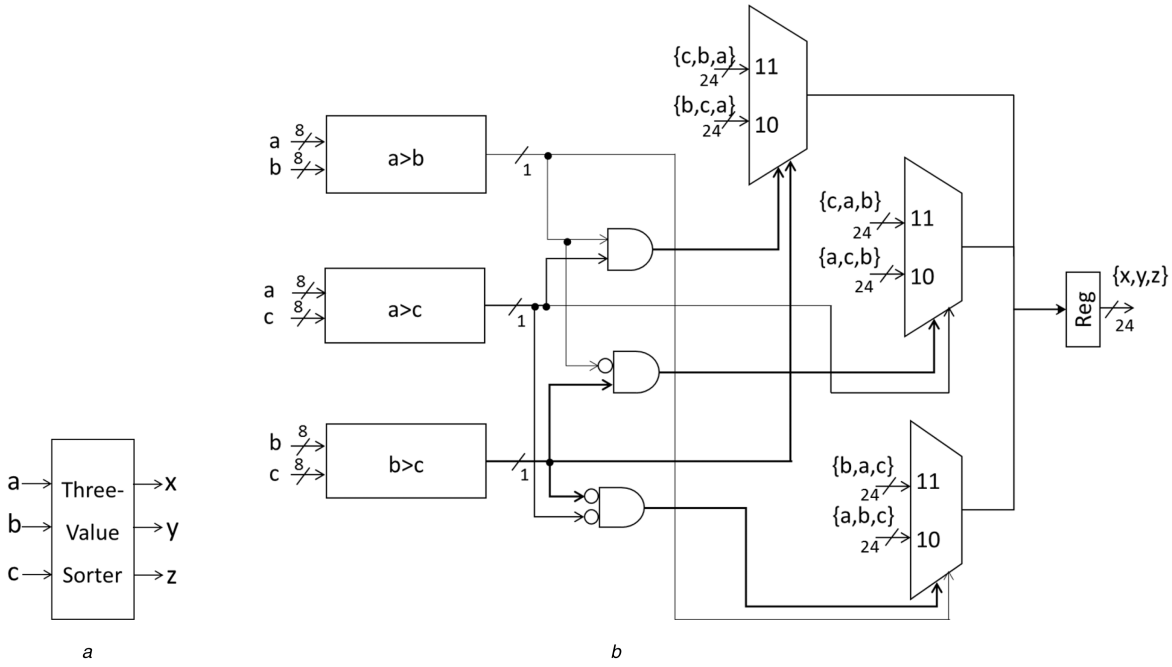


Fig. 8 *Conventional three-value sorter [21]*
(a) Three-value sorter, ***(b)*** Two-value sorter used in (a)



a *b*

Fig. 9 *Proposed three-value sorter*
(a) Sorting module, ***(b)*** Circuit diagram

Table 1 Synthesis results of the proposed 5 × 5 median filter (sliding window architecture and LLMF core) for Zynq xc7z020-1clg484

Device utilisation summary			
Logic utilisation	Used	Available	Utilisation, %
number of slice registers	1550	106,400	1
number of slice LUTs	1706	53,200	3
number of block RAMs	2	140	1
maximum frequency, f_{max} = 394 MHz			

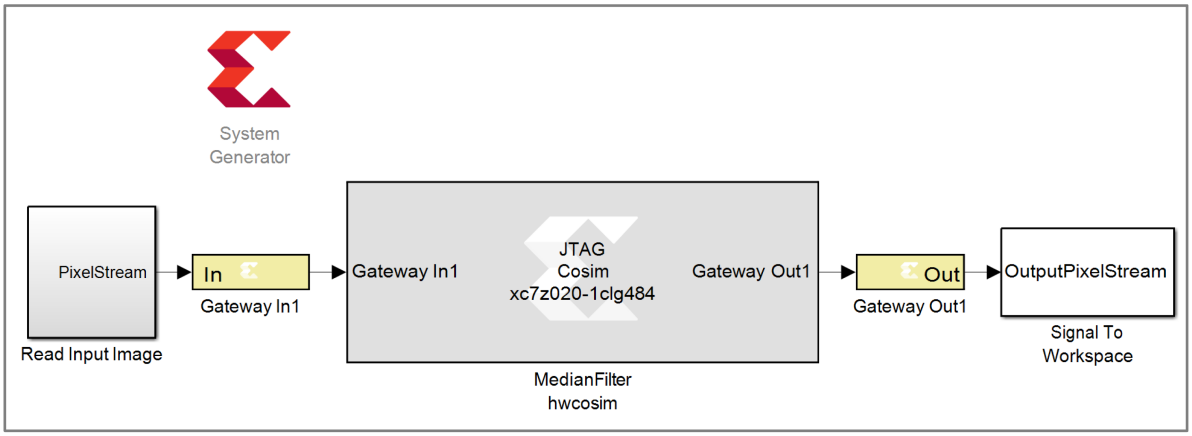


Fig. 10 *Set-up to test the proposed median filter hardware on FPGA*

can also be used as two independent 1 K × 16-bits dual port RAMs. Therefore, two block RAMs are utilised for implementing four line buffers of size 640 each, as shown in Table 1. Table 1 shows FPGA resources consumed by complete 5 × 5 median filter architecture including both LLMF core and sliding window architecture. However, resources consumed by LLMF core alone are shown in Table 4. The slice registers are utilised mainly for implementing the pipelined registers used in the architecture and sorting circuits consume the slice LUTs.

4.2 Performance results

For testing the median filter hardware on FPGA, the 'Xilinx's system generator for DSP' tool of VIVADO v2014.4 was used. A Simulink model was created, which uses Simulink blocksets for reading image files from computer's hard disk and saving the output image into MATLAB's workspace. A snapshot of Simulink model is shown in Fig. 10, which was used for testing the proposed median filter hardware on ZedBoard (Zynq FPGA board). The output pixel stream is saved as one-dimensional array of pixels and it is viewed as image using MATLAB. The input and output images are shown in Fig. 11.

The 'ordfilt2 ()' is the built-in function of MATLAB for median filtering. The output image obtained from this function exactly (100%) matches with the output image obtained from the proposed median filter hardware.

The initial clock cycle latency of the proposed median filter architecture is shown in Table 2 for an image of 640 × 480 pixels. The latency of 3 of the proposed 5 × 5 LLMF core is obtained as follows. Referring back to Fig. 5*c*, there are four stages in the

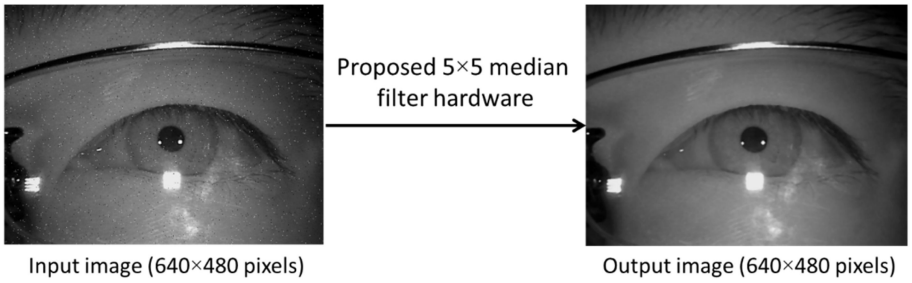


Fig. 11 *Image filtering using the proposed median filter*

Table 2 Clock cycle latency of the proposed median filter architecture

Module	Clock cycle latency (for 640 × 480 pixel image, WIDTH = 640)
5 × 5 sliding window	2565 (=4WIDTH + 5)
5 × 5 LLMF core	3
overall median filter	2568

Table 3 Processing time per image of the proposed median filter hardware

Image size (pixels)	Number of clock cycles	Processing time
320 × 240	$1288 + 320 \times 240$	197.56 μs
640 × 480	$2568 + 640 \times 480$	783.72 μs
1280 × 720	$5128 + 1280 \times 720$	2.34 ms

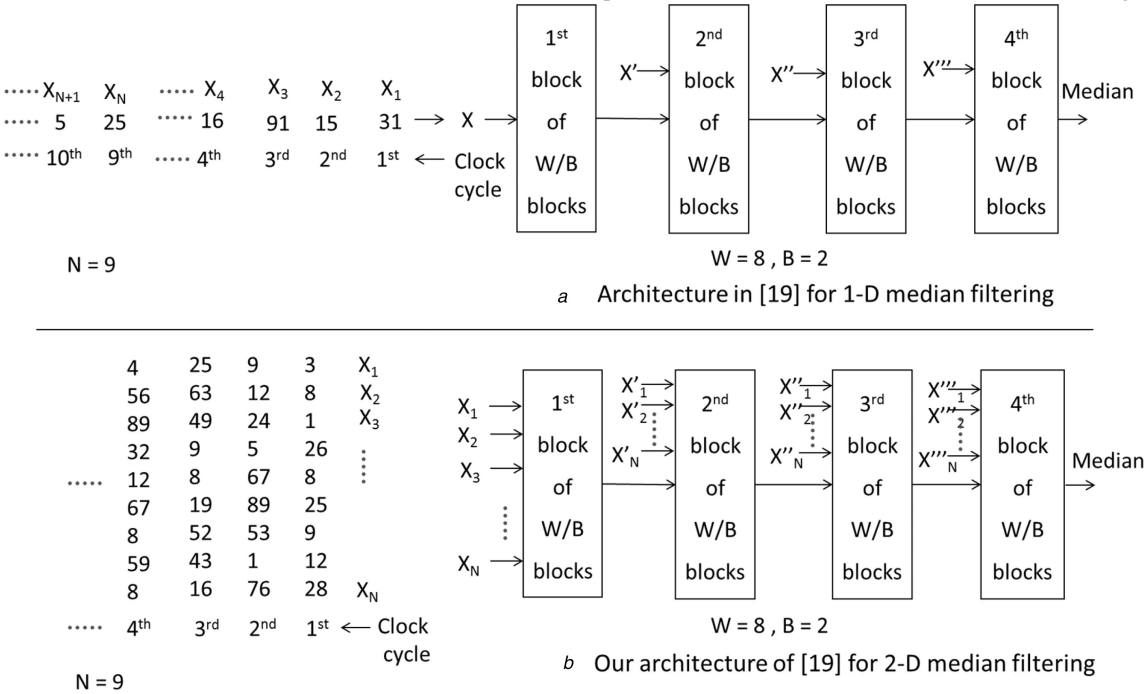


Fig. 12 *Difference between the architecture presented in [19] and the one that was implemented here using [19] algorithm*
(a) Architecture in [19], ***(b)*** Our architecture of [19]

architecture and between these stages, there are three pipeline cut in the form of pipelined registers. Each of the first three stages takes a single clock cycle for computing and storing the result in pipelined registers. The fourth stage is composed of a single three-value sorter, which is a combinational circuit and does not require clock. Therefore, latency is equal to the pipeline depth that is 3.

The total number of clock cycles the architecture takes for completing the median filtering task is shown in Table 3. The processing time is obtained by multiplying the number of clock cycles and clock period (f_{max} = 394 MHz) as shown in Table 3.

4.3 Comparison with other architectures

In order to compare the proposed work with previous work, we have implemented on the same FPGA device the median calculation algorithm presented in [19]. We choose the implementing [19] method over [3], as it can compute the median in *W/B* steps, whereas [3] takes *W* steps for *W*-bit integers. In [19], the implementation results show that [19] is better than [3] in terms of latency, maximum frequency, FPGA resources, and design flexibility. We did not implement the same hardware architecture presented in [19], because this architecture was designed for one-dimensional (1D) median filtering, where a serial integer-stream (one integer per clock cycle) is fed to the architecture and two consecutive samples of *N* integers each differ by one integer and the rest of $N-1$ integers are the same as shown in Fig. 12*a*. Since this paper focuses on two-dimensional (2D) median filtering, we have redesigned 1D median filter architecture of [19] in order to perform 2D median filtering as described below.

4.3.1 Our hardware implementation for 2D median filtering using the [19] algorithm: We implemented dedicated hardware for the median calculation algorithm presented in [19] in order to perform 2D median filtering. This hardware architecture shown in Fig. 12*b* is fed with a set of *N* integers per clock cycle and two consecutive sets (samples) of *N* integers each need not to have common integers in them, as opposed to the architecture of [19] shown in Fig. 12*a*. The architecture finds the median of *N W*-bit integers using *W/B* processing blocks (stages), where $W = 8$ and $B = 2$. The latency of this architecture is $2(W/B)-1 = 7$ clock cycles, where each of *W/B* blocks has a pipeline depth of 2 except last one, which has 1. Therefore, the overall pipeline depth of this architecture is 7. The implementation results are shown in Table 4.

Table 4 Comparison of median filter architectures (results are for the same FPGA device, Zynq xc7z020-1clg484)

	3 × 3 core ($N = 9$)		5 × 5 core ($N = 25$)	
	[19]$^a B = 2$	Proposed	[19]$^a B = 2$	Proposed
slice registers	195	192	421	1304
slice LUTs	594	203	2873	1650
max. frequency, f_{max} (MHz)	280	516	152	394
clock cycle latency	7	2	7	3

[a]This is our hardware implementation for 2D median filtering using the [19] algorithm.

Table 5 Comparison of the conventional [21] and the proposed three-value sorter (results are for the same FPGA device, Zynq xc7z020-1clg484)

Three-value sorter	Slice LUTs	f_{max} of 3 × 3 core, MHz	f_{max} of 5 × 5 core, MHz
conventional [21]	57	258	66
proposed	39	516	394

We started with $B = 2$ for our convenience as [19] also presented procedure and results for $B = 2$, but not for $B = 3$ or 4 with $N = 25$.

We implemented [19] with $B = 2$ as shown in Table 4, and having looked at the results, we did not go for $B = 3$, as it would further reduce the frequency, but would give improved latency of $2(W/B)-1 = 5$ clock cycles [19]. Moreover, going from $B = 2$ to $B = 3$ will further increase FPGA resources, because 2^B-bit codes are generated by binary to thermometer encoding [19] or in other words, 2^B counters along with their update logic are required.

It is seen from Table 4 that the proposed architecture shows better results than [19] in terms of latency, maximum frequency, and FPGA slice LUTs. The latencies of the proposed architectures are better than [19] as shown in Table 4. Moreover, latency of the proposed architecture is independent of W (bit-length) of integers, whereas with increase in W, the latency in [3, 19] will also increase, as they are W/B-step and W-step methods, respectively. The process in [19] is slower than the proposed architecture due to their different critical paths as follows. The critical path in [19] consists of a sequential N-step process of parallel counter update done accumulatively by encoding bits inside a W/B processing block [19, 22], whereas the critical path in the proposed 5 × 5 LLMF core is 'sorting of five values', which is a sequential four-step process of sorting of three values. Table 4 also shows that the maximum frequency of [19] falls down drastically when moving to $N = 25$ from $N = 9$, which happens due to the delay caused by aforementioned sequential N-step process (critical path). In the proposed architectures, 3 × 3 core is faster than 5 × 5 core, because its critical path consists of a single three-value sorter. However, 5 × 5 core can be made faster by introducing one or more pipeline cuts in its critical path (that is, sorting of five values), but this would increase the latency by 3 per pipeline cut, as three stages of the architecture are using sorting of five values. This gives a designer the flexibility to trade-off easily the increase in frequency at the cost of latency. Note that the proposed 3 × 3 core architecture implements the algorithm depicted in Fig. 2 using the proposed three-value sorter and the same architectural approach that was used for the proposed 5 × 5 LLMF core.

4.3.2 Extension to the higher window sizes: In the proposed architectures, the FPGA resources increase drastically when moving from 3 × 3 core to 5 × 5 core as shown in Table 4, which impose limitation on going for higher window size, such as 7 × 7 or 9 × 9. Moreover, frequency and latency would also degrade for higher window sizes. However, these degradations in terms of latency, frequency, and FPGA resources utilisation also do happen with [19] as evident from Table 4.

Table 5 compares the conventional and the proposed three-value sorters (Figs. 8 and 9). It is evident from Table 5 that the proposed three-value sorter is faster than the conventional three-value sorter [21], as it drastically improves the overall maximum frequency (f_{max}) of operation of the complete architecture. It also consumes lesser number of slice LUTs than conventional sorter.

5 Conclusion

This paper has presented a low-latency filter core for hardware implementation of 5 × 5 median filter, for image and video processing applications. Parallel and pipelined implementation techniques were used, which gives throughput of one pixel per clock cycle for the median filtering of greyscale images. This paper also has presented a three-value sorter, which is faster than conventional sorter, and a method for sorting of higher values using three-value sorter. The processing time of the proposed median filter is 783.72 µs for an image of 640 × 480 pixels. The proposed median filter hardware can be used to meet real-time performance in embedded applications, where low speed and low-cost CPUs are used. In computer vision applications, such as personal surveillance systems, the real-time performance is often understood as the response time to be in order of milliseconds and sometimes microseconds. This performance constraint is met by our architecture as evident from the processing time results. The presented dedicated hardware unit for median filtering can be used in hardware–software co-design-based embedded system development on FPGA, where compute-intensive tasks are implemented as dedicated hardware units to offload CPU core.

6 References

[1] Davies, E.R.: '*Computer and machine vision: theory, algorithms, practicalities*' (Academic Press, New York, 2012)

[2] Kumar, V., Asati, A., Gupta, A.: 'Accurate iris localization using edge-map generation and adaptive circular Hough transform for less constrained infrared iris images', *Int. J. Electr. Comput. Eng.*, 2016, **6**, (4), pp. 1637–1646

[3] Prokin, D., Prokin, M.: 'Low hardware complexity pipelined rank filter', *IEEE Trans. Circuits Syst. II Express Briefs*, 2010, **57**, (6), pp. 446–450

[4] Bailey, D.G.: '*Design for embedded image processing on FPGAs*' (John Wiley & Sons (Asia) Pvt. Ltd, Singapore, 2011)

[5] Lee, T.W., Lee, J.H., Choo, S.B.: 'FPGA implementation of a 3 × 3 window median filter based on a new efficient bit-serial sorting algorithm'. Proc. 7th KORUS, June–July 2003, pp. 237–242

[6] Chang, L.W., Lin, J.H.: 'A bit-level systolic array for median filter', *IEEE Trans. Signal Process.*, 1992, **40**, (8), pp. 2079–2083

[7] Benkrid, K., Crookes, D., Benkrid, A.: 'Design and implementation of a novel algorithm for general purpose median filtering on FPGAs'. Proc. IEEE Int. Symp. on Circuits and Systems, May 2002, pp. 425–428

[8] Fahmy, S.A., Cheung, P.Y.K., Luk, W.: 'Novel FPGA-based implementation of median and weighted median filters for image processing'. Proc. Int. Conf. on Field Programmable Logic and Applications, August 2005, pp. 142–147

[9] Vasicek, Z., Sekanina, L.: 'Novel hardware implementation of adaptive median filters'. Proc. 11th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, April 2008, pp. 1–6

[10] Scott, J., Pusateri, M., Mushtaq, M.U.: 'Comparison of 2D median filter hardware implementations for real-time stereo video'. Proc. 37th IEEE Applied Imagery Pattern Recognition Workshop, October 2008, pp. 1–6

[11] Batcher, K.E.: 'Sorting networks and their applications'. Proc. Joint Computer Conf. – AFIPS, April–May 1968, p. 307

[12] Thompson, C.D.: 'The VLSI complexity of sorting', *IEEE Trans. Comput.*, 1983, **C-32**, (12), pp. 1171–1184

[13] Danielsson, P.E.: 'Getting the median faster', *Comput. Graph. Image Process.*, 1981, **17**, (1), pp. 71–78

[14] Knuth, D.E.: '*The art of computer programming volume 3: sorting and searching*', vol. **3** (Addison Wesley, Reading, MA, 1998), p. 829

[15] Herzog, S.: 'Efficient DSP implementation of median filtering for real-time audio noise reduction'. Proc. of the 16th Int. Conf. on Digital Audio Effects, Maynooth, Ireland, September 2013, pp. 1–6

[16] Lu, X., Song, L., Shen, S., *et al.*: 'Parallel Hough transform-based straight line detection and its FPGA implementation in embedded vision', *Sensors*, 2013, **13**, (7), pp. 9223–9247

[17] Sarawadekar, K.P., Indana, H.B., Bera, D., *et al.*: 'VLSI-DSP based real time solution of DSC-SRI for an ultrasound system', *Microprocess. Microsyst.*, 2012, **36**, (1), pp. 1–12

[18] Kumar, V., Asati, A., Gupta, A.: 'Hardware implementation of a novel edge-map generation technique for pupil detection in NIR images', *Eng. Sci. Tech. Int. J.*, 2016, **20**, (2), pp. 694–704 http://dx.doi.org/10.1016/j.jestch.2016.11.001

[19] Cadenas, J.: 'Pipelined median architecture', *Electron. Lett.*, 2015, **51**, (24), pp. 1999–2001

[20] Vasanth, K., Karthik, S., Nirmal-Raj, S., *et al.*: 'FPGA implementation of optimized sorting network algorithm for median filters'. Proc. Int. Conf. on Emerging Trends in Robotics and Communication Technologies, December 2010, pp. 224–229

[21] Meena, S.M., Linganagouda, K.: 'Implementation and analysis of optimized architectures for rank order filter', *J. Real-Time Image Process.*, 2008, **3**, (1–2), pp. 33–41

[22] Cadenas, J.O., Megson, G.M., Sherratt, R.S.: 'Median filter architecture by accumulative parallel counters', *IEEE Trans. Circuits Syst. II Express Briefs*, 2015, **62**, (7), pp. 661–665