Sorting Network Based Architectures for Median Filters

Chaitali Chakrabarti

Department of Electrical Engineering
Center for Solid State Electronics Research
Arizona State University
Tempe, AZ 85287-5706
(602) 965-9516
email: chaitali@enuxha.eas.asu.edu

Abstract

This paper presents sorting network based architectures for computing non-recursive and recursive median filters. The proposed architectures are highly pipelined, and consist of fewer compare-swap units than existing architectures. The reduction in the number of compare-swap units is achieved by minimizing computational overlap between successive outputs, and also by using Batcher's odd-even merge sort (instead of bubble-sort). The latency of these networks is reduced by building them with sorting units that sort 2 elements (*sort-2*) as well as 3 elements (*sort-3*) in 1 time unit.

1 Introduction

Median filter is a non-linear digital filter with an interesting combination of properties: it can smooth the signal by removing noise, and at the same time preserve edge information. These properties make it a very popular filter in speech processing and image processing schemes.

There are two types of median filters: non-recursive and recursive. In non-recursive median filtering, a window is moved along the sampled values of the image, and the center value of each window is replaced by the median of the values in the window. For instance, in 2-D non-recursive median filtering, the (i, j)th window of size $(K \times K)$, $W_{i,j}$, is centered at (i, j), and the (i, j)th output $y_{i,j} = median\{W_{i,j}\}$. In recursive median filtering, the window consists of recent median values as well as the input values. In 1-D recursive median filtering, the ith window of size $(2N + 1)$, $\tilde{W}_i$, consists of $(N + 1)$ input values $x_i, \ldots, x_{i+N}$, and N output values $y_{i-N}, \ldots, y_{i-1}$; $\tilde{W}_i = \{y_{i-N}, \ldots, y_{i-1}, x_i, \ldots, x_{i+N}\}$, and the ith output $y_i = median\{\tilde{W}_i\}$.

The existing architectures for median filters can be broadly classified into two classes: array based architectures [6, 8, 7], and sorting network based architectures [11, 9, 10]. [12] gives an excellent survey of the existing architectures. The array based architectures consist of M processors if the window is of size M, but they have a large sample period compared to sorting network based architectures. Recently we showed how to reduce the sample period of such architectures by pipelining the computations in each processor [4]. The sorting network based architectures are inherently pipelined, but they consist of a large number of compare-swap[1] units. Most of the existing architectures in this class [11, 9] are based on bubble-sort, and require $M\lfloor M/2 \rfloor$ compare-swap units to sort M elements. In [5] we developed a median filter architecture based on Batcher's bitonic sort [2] which reduced the number of compare-swap units to $O(M \log^2 M)$. Bitonic sort is based on merging sequences of equal sizes, and as a result cannot be used to efficiently sort windows of all sizes. Batcher's odd-even merge sort, on the other hand, does not have any restriction on the size of the merging sequences, and is highly suitable for sorting windows of all sizes.

In this paper we develop sorting network based architectures based on Batcher's odd-even merge sort [2] for both non-recursive and recursive median filters. The proposed architectures are highly pipelined and have fewer compare-swap units than existing architectures [9, 11, 3]. We reduce the number of compare-swap units further by minimizing the number of overlapping computations among consecutive windows, and also by processing consecutive windows at the same time. When M is not a power of 2, we reduce the the latency and increase the regularity of the sorting network by building it with sorting units that sort 2 elements as well as 3 elements in 1 time unit. In the next two sections,

[1] A compare-swap unit compares two elements and sorts them in decreasing or increasing order.

we describe sorting network-based architectures for non-recursive and recursive median filters.

2 Non-recursive median filtering

In this section we describe two sorting network architectures that are based on Batcher's odd-even merge sort [2] for non-recursive median filtering. The first network sorts *all* the elements of a window from scratch every sample period, while the second network exploits the overlap between consecutive windows, and sorts a subset of the window elements from scratch every sample period.

Batcher's odd-even merge sort: The odd-even merge algorithm to sort M elements [2] consists of parallely sorting the top $\lceil M/2 \rceil$ elements and the bottom $\lfloor M/2 \rfloor$ elements in decreasing (or increasing) order, and then merging the two sorted sequences. When M is not a power of 2, the latency of the odd-even merge network can be reduced by introducing sorting units which sort 3 elements in 1 time unit. Such a method of increasing circuitry to reduce latency could be extended to sort any number of elements in 1 time unit. However the circuitry required to support such an operation would increase drastically with increasing M. The number of 2-element comparators in this sorting network is given by

$$C_{sort}(M) = C_{sort}(\lceil M/2 \rceil) + C_{sort}(\lfloor M/2 \rfloor) + C_{merge}(\lceil M/2 \rceil, \lfloor M/2 \rfloor),$$

where $C_{merge}(\lceil M/2 \rceil, \lfloor M/2 \rfloor)$ is the number of comparators required to merge two sorted sequences of sizes $\lceil M/2 \rceil$ and $\lfloor M/2 \rfloor$.

$$C_{merge}(\lceil M/2 \rceil, \lfloor M/2 \rfloor) = C_{merge}(\lceil \frac{\lceil M/2 \rceil}{2} \rceil, \lceil \frac{\lfloor M/2 \rfloor}{2} \rceil) + C_{merge}(\lfloor \frac{\lceil M/2 \rceil}{2} \rfloor, \lfloor \frac{\lfloor M/2 \rfloor}{2} \rfloor) + \lceil M/2 \rceil - 1,$$

$C_{merge}(1,1) = 1$, $C_{merge}(2,1) = 2$ and $C_{sort}(2) = 1$.

Network 1: This network is obtained from the odd-even merge sorting network by pruning the last merge unit (which merges sequences of sizes $\lceil M/2 \rceil$ and $\lfloor M/2 \rfloor$). The number of 2-element comparators in this network is given by

$$C_{med}(M) = C_{sort}(\lceil M/2 \rceil) + C_{sort}(\lfloor M/2 \rfloor) + C_{med}(\lceil M/2 \rceil, \lfloor M/2 \rfloor), \qquad \text{where}$$

$$C_{med}(\lceil M/2 \rceil, \lfloor M/2 \rfloor) = C_{med}(\lceil \frac{\lceil M/2 \rceil}{2} \rceil, \lceil \frac{\lfloor M/2 \rfloor}{2} \rceil) + C_{med}(\lfloor \frac{\lceil M/2 \rceil}{2} \rfloor, \lfloor \frac{\lfloor M/2 \rfloor}{2} \rfloor) + 1,$$

$C_{med}(1,1) = 1$ and $C_{med}(2,1) = 2$. Figure 1a describes Network 1 for computing the median of 9 elements. This network requires 22 2-element comparators compared to 36 2-element comparators in [11] and 30 2-element comparators in [9]. The number of stages in this network is 6 compared to 9 stages in the networks of [9] and [11].

Network 2: This network is based on the fact that in 2-D median filtering, the window shifts by 1 column every sample period, and that two consecutive windows share $(K-1)$ columns. In this network, the elements of a column are sorted only once, and the sorted column is stored for $(K-1)$ cycles, so that it can be used for the computation of K consecutive outputs. Thus the K columns of a $(K \times K)$ window are sorted during K different cycles, and the sorted columns are merged using Batcher's odd-even merge sort for every median output. Figure 1b describes Network 2 for computing the median of a (3×3) window. The number of 2-element comparators is only 16.

We next describe the sorting units that are used to build the two median networks.

Sort-2 unit: The *sort-2* unit (see Figure 2a) is the conventional compare-swap unit. The comparator compares the two inputs A and B, and generates a control signal X; $X = 1$ if $A > B$, and is 0 otherwise. The output logic computes the functions $max(A,B) = AX + B\bar{X}$, and $min(A,B) = A\bar{X} + BX$.

Sort-3a unit: The *sort-3a* unit (see Figure 2b) sorts 3 unordered elements in 1 time unit. The 3 comparators compare the pairs (A,B), (B,C), and (A,C), and generate control signals X, Y, and Z; $X = 1$ if $A > B$ and is 0 otherwise; $Y = 1$ if $B > C$ and is 0 otherwise; $Z = 1$ if $C > A$ and is 0 otherwise. The output logic computes the functions $max(A,B,C) = AX\bar{Z} + B\bar{X}Y + C\bar{Y}Z$, $med(A,B,C) = A(XZ + \bar{X}\bar{Z}) + B(XY + \bar{X}\bar{Y}) + C(YZ + \bar{Y}\bar{Z})$, and $min(A,B,C) = A\bar{X}Z + BX\bar{Y} + CY\bar{Z}$. The *sort-3a* unit is the most complex of these three units. The delay in the circuitry of this unit determines the clock period of the network.

Sort-3b unit: The *sort-3b* unit (see Figure 2c) is used to merge two ordered sequences of sizes 2 and 1 in 1 time unit. Let (A,C) be an ordered sequence to be merged with B, $A > C$. The two comparators generate two control signals X and Y; $X = 1$ if $A > B$ and is 0 otherwise; $Y = 1$ if $B > C$ and is 0 otherwise. The output logic computes the functions $max(A,B,C) = \bar{X}B + XA$, $med(A,B,C) = \bar{X}YA + XYB + X\bar{Y}C$, and $min(A,B,C) = \bar{Y}B + YC$.

A (3×3) median filter operating in bit-serial mode, with least significant bit (LSB) first, and 8-bit precision, was designed using VTITools on an Apollo workstation. The clocking scheme was 2-phase non-overlapping. The technology was 2 micron twin-tub CMOS. According to the simulations that were run using Berkeley SPICE and VTISim, this network can be clocked at 50 MHz, resulting in a throughput of 6×10^6 medians/second.

2.1 Block processing

The number of comparators per output can be reduced even more by processing consecutive windows at the same time. This is because consecutive windows have common elements, and so if they are processed simultaneously, the common elements need to be sorted only once. Processing multiple outputs at the same time (block processing) has many advantages including, reduction in power dissipation per output[2] and fast processing of stored signals. In this section, we first develop efficient ways of block processing 1-D median filters, and then show how this can be extended to 2-D filters.

In [10], Lucke and Parhi developed a block processing algorithm which identifies the elements that are common to all the outputs, and also identifies pairs of elements that are common to more than one output. The common elements are presorted and then merged to generate all the block outputs. Here we propose a new algorithm for 1-D block processing that is based on merging sequences of approximately equal size in every stage. Our algorithm differs from [10] in the way the common elements are identified, as well as in the way the sorted sequences are merged. Our algorithm is summarized as follows. Let the 1-D window be of size $K = 2N + 1$, and the block be of size B.

Step 1: Identify disjoint groups of $N + 1$ consecutive elements which occur with the maximum frequency. Sort these groups of $N + 1$ elements.

This involves (i) creating groups of $(N + 1)$ consecutive elements from the list of B outputs, and listing the frequency of occurrence of each group, (ii) forming sets of such groups, such that the groups in a set do not contain any common elements, and that they occur with high frequency, (iii) choosing the set which contains minimum number of groups, and in case of conflict, choosing the set whose groups occur approximately same number of times, (iv) sorting the $(N + 1)$ elements of a group by odd-even merge sort as described in Section 2.

Step 2: Sort the remaining N elements per output.

The remaining N elements for every output are sorted by applying this algorithm recursively.

Step 3: Merge sorted sequences of sizes $N + 1$ elements (from Step 1) and N elements (from Step 2).

Since we are interested only in the median, this merge unit is modified as described in Section 2.

The above algorithm is used to generate a network to compute multiple median outputs, referred to as the multiple-median network. The number of comparators in such a network can be reduced by identifying common elements between the $(N+1)$ elements of a group in Step 1, and the N elements of a group in Step 2. For instance, if $\{x_3, x_4, x_5, x_6\}$ is one of the groups in Step 1, and if $\{x_2, x_3, x_4\}$ is one of the groups in Step 2, then $\{x_3, x_4\}$ which is common to both the groups is presorted, and then merged once with $\{x_5, x_6\}$ and once with $\{x_2\}$ to generate the two sorted groups. Another

[2] In block processing, if B outputs are computed every cycle, then the power dissipation is proportional to the number of comparators in the network, since all the comparators are active. If B outputs are computed every B cycles, then the power dissipation is proportional to the number of comparators that are active in every Bth stage of the network (the remaining comparators are deactivated). In both cases, the power dissipation per output is less, at the expense of a larger number of comparators on chip.

way of reducing the number of comparators in the multiple-median network is to identify groups of elements whose indices differ by B units, and to sort only the groups with larger indices. For instance, if the multiple-median network has groups $\{x_i, x_{i+1}\}$, and $\{x_{i+B}, x_{i+B+1}\}$, then $\{x_i, x_{i+1}\}$ need not be sorted, since the sorted sequence corresponding to $\{x_i, x_{i+1}\}$ is the sorted sequence $\{x_{i+B}, x_{i+B+1}\}$ delayed by B units.

Figure 3 describes the derivation of a multiple median network when $K = 9$, and $B = 3$. The number of sort-2 units is 48 compared to $22 \times 3 = 66$ units that would be required if there were no block processing. Figure 4a compares the number of comparators required by the proposed method with those in [10] for various values of K and B. Figure 4b shows how for the same value of K, the number of comparators/output vary with different values of B.

The algorithm for 1-D block processing can be easily extended to 2-D by replacing each element of the 1-D window by a column (or row) of $K = 2N + 1$ sorted elements. Merging two sorted lists of sizes $\lceil M/2 \rceil$ and $\lfloor M/2 \rfloor$ in 1-D translates to merging two sorted sequences of sizes $\lceil M/2 \rceil K$ and $\lfloor M/2 \rfloor K$ in 2-D. Batcher's odd-even merge sort is used to merge these sequences. For median computation, the last merge unit (involving sequences of sizes $\lceil K/2 \rceil K$ and $\lfloor K/2 \rfloor K$) is pruned by a method similar to that described in Section 2.

3 Recursive median filtering

The sorting network based architecture for recursive median filtering consists of a sorting network to sort the $(N+1)$ input values using Batcher's odd-even merge sort, followed by N insertion stages to insert N output values one at a time, into the sorted list. For instance, the sorted list of W_i is built by inserting y_{i-N} in stage 1, y_{i-N+1} in stage 2, and finally y_{i-1} in stage N. Figure 5a gives the block diagram of this architecture. At any instant of time, each of these stages operates on a different window. For instance, during the computation of y_i, Stage 1 operates on window W_{i+N-1}, Stage 2 on window W_{i+N-2}, and Stage N on window W_i. Since y_i is an element of windows W_{i+N-1} through W_{i+1}, it is fed back to each of the N insertion stages.

Design of insertion stage S_j: In insertion stage S_j, $1 \leq j \leq N$, the output value y is inserted in a sorted sequence $u_{j,1}, \ldots, u_{j,N+j}$, to create another sorted sequence $u_{j+1,1}, \ldots u_{j+1,N+j+1}$. The hardware consists of $(N + j)$ comparators followed by output logic units as shown in Figure 5b. The ith comparator compares $u_{j,i}$ with y and sets $A_i = 1$ if $u_{j,i} > y$, $1 \leq i \leq N + j$. A_i is fed to the output logic units of $u_{j+1,i}$ and $u_{j+1,i+1}$. There are three types of output logic units. Output logic unit 1 outputs the maximum of the two inputs $u_{j,1}$ and y; $u_{j+1,1} = u_{j,1}A_1 + y\bar{A}_1$. Output logic unit 2 outputs the median of the three inputs $u_{j,l-1}$, $u_{j,l}$, and y, $2 \leq l \leq N + j$;

$u_{j+1,l} = \bar{A}_{l-1}\bar{A}_l u_{j,l-1} + (\bar{A}_{l-1}A_l + A_{l-1}\bar{A}_l)y + A_{l-1}A_l u_{j,l}$. Output logic unit 3 outputs the minimum of the two inputs $u_{j,N+j+1}$ and y; $u_{j+1,N+j+1} = u_{j,N+j}\bar{A}_{N+j} + yA_{N+j}$.

The number of comparators in the N insertion stages is $(N+1)+(N+2)+\ldots+2N = N(3N+1)/2$. In comparison, in the architecture of Boncelet [3], S_j is built with $(N+j+1)$ MAX-MIN cells, each of which computes the median of 3 inputs using 2 comparators. The total number of comparators in the realization of [3] is $3N(N+1)$.

The number of comparators in the N insertion stages can be reduced further if the rank of the filter is fixed. The reduction in the number of comparators is achieved by eliminating all comparators which do not contribute to the computation of a particular ranked output. For instance, for median filters, stage S_N requires only 2 comparators, since the median output is a function of $u_{N,N}$, $u_{N,N+1}$ and y, stage S_{N-1} requires only 3 comparators, since the median output is a function of $u_{N-1,N-1}$, $u_{N-1,N}$, $u_{N-1,N+1}$ and y, and so on. The total number of comparators in the N insertion stages of a median filter is reduced to $(N+1)+N+\ldots+2 = N(N+3)/2$. The number of comparators in the sorting network for the $(N+1)$ input samples can also be reduced by noting that two consecutive windows have N elements in common. The $(N+1)$ element sorting network can be replaced by an array of $2N$ comparators (to compare with the old and the new elements) and logic units to correctly gate the outputs of the comparisons as in [1].

The sample period of this architecture is the delay in the insertion stage. For bit-serial operation, the sample period is the time required to insert an element into a sorted list. Reducing the sample period by a factor of M involves restructuring the algorithm such that y_i is a function of $y_{i-M}, y_{i-2M}, \ldots, y_{i-(N-1)M}$. This problem is currently under investigation. The sample period of a bit-parallel realization with most significant bit (MSB) first operation, can be reduced by any factor by applying Karaman's method [9] to pipeline the insertion stages.

References

[1] G. R. Arce, P. J. Warter, "A median filter architecture suitable for VLSI implementation," Proc. of 23rd Annual Allerton Conf. on Comm., Control, and Computing, 1984, pp. 172-181.

[2] K. E. Batcher, "Sorting network and their applications," 1968 Spring Joint Computer Conference, AFIPS Proc. vol. 32, pp. 307-314.

[3] C. G. Boncelet, "Recursive Algorithms and VLSI Implementations for Median Filtering," Proc. IEEE Int. Conf. Circuits and Systems, pp. 1745-1747, 1988.

[4] C. Chakrabarti, "High Sample Rate Array Architectures for Median Filters," accepted for publication in IEEE Trans. on Signal Processing.

[5] C. Chakrabarti, S. Dhanani, "Median Filter Architecture based on Sorting Networks," Proc. IEEE Int. Conf. Circuits and Systems, 1992.

[6] A. Fisher, "Systolic algorithms for running order statistics in signal and image processing," J. Digital Syst., vol. 4, 1982, pp. 251-264.

[7] J. N. Hwang, J. M. Hong, "Systolic architectures for 2-D rank order filtering," Proc. Int. Conf. Application Specific Array Processors (Princeton, NJ), 1990, pp. 90-99.

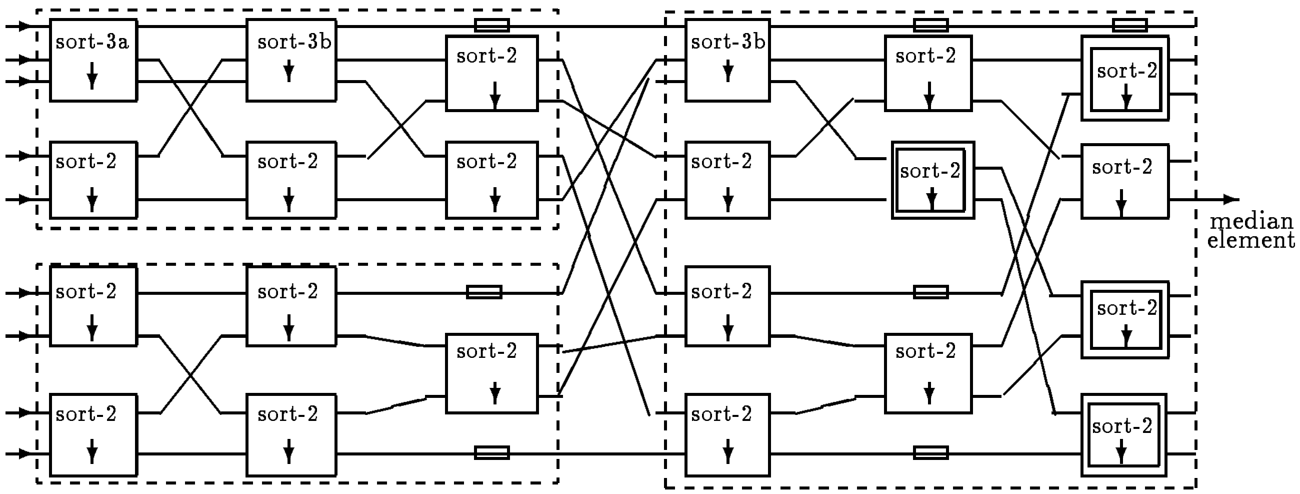
[8] S. Y. Kung, "VLSI Array processors," Prentice Hall, 1989.

[9] M. Karaman, L. Onural, A. Atalar, "Design and implementation of a general purpose median filter in VLSI," VLSI Signal Processing III, 1988, pp. 111-119.

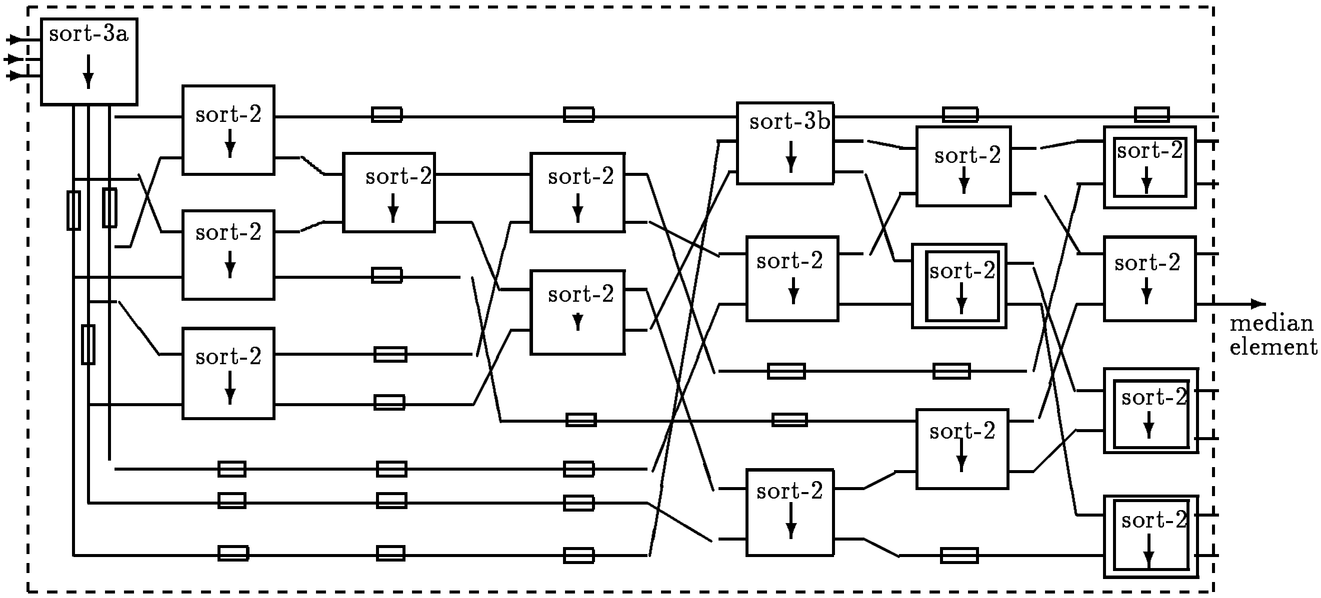
[10] L. Lucke, K. Parhi, "Parallel Structures for Rank Order and Stack Filters," Proc. IEEE Int. Conf. Circuits and Systems, 1992.

[11] K. Oflazer, "Design and implementation of a single chip 1-D median filter," IEEE Trans on Acoust., Speech, and Signal Processing, vol. ASSP-30, 1983, pp. 1164-1168.

[12] D. Richards, "VLSI median filters," IEEE Trans on Acoust., Speech, and Signal Processing, vol. 38, 1990, pp. 145-153.



(a)



(b)

Figure 1: (a) Network 1 for computing the median of 9 elements. The network is obtained by eliminating 4 sort-2 units (double boxed) from the merge unit of the corresponding sorting network. The number of comparators is 22. The flat boxes represent latches. (b) Network 2 for computing the median of a (3×3) window. The elements of a column are sorted only once. The number of comparators is only 16.

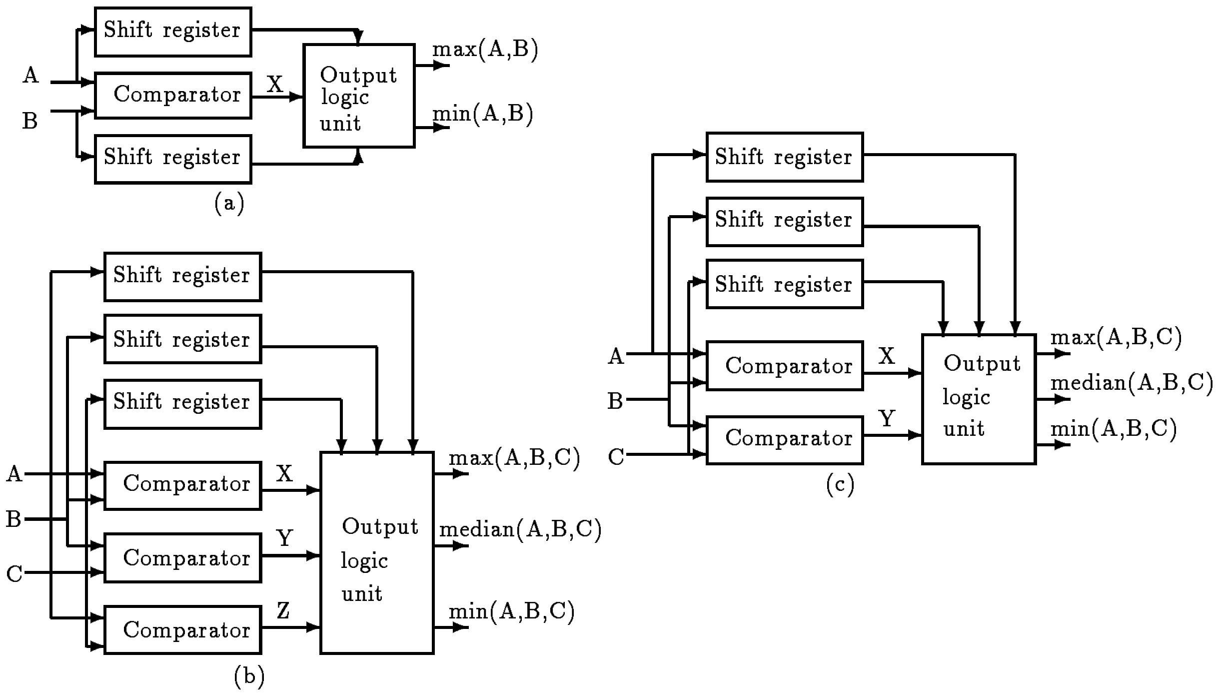


Figure 2: Block diagrams of (a) sort-2 unit (b) sort-3a unit and (c) sort-3b unit

K=9 $\quad y_1 = med\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$

B=3 $\quad y_2 = med\{x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

$\quad y_3 = med\{x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\}$

Step 1: Groups $\{x_4, x_5, x_6, x_7, x_8\}$ and $\{x_5, x_6, x_7, x_8, x_9\}$ occur with maximum frequency. Choose $S = \{x_4, x_5, x_6, x_7, x_8\}$.

Step 2: Remaining elements: $\{x_1, x_2, x_3, x_9\}$, $\{x_2, x_3, x_9, x_{10}\}$, $\{x_3, x_9, x_{10}, x_{11}\}$.
Group $\{x_3, x_9\}$ occurs with maximum frequency.
Remaining elements $\{x_1, x_2\}$, $\{x_2, x_{10}\}$, $\{x_{10}, x_{11}\}$. Note that the sorted sequence corresponding to $\{x_1, x_2\}$ is obtained by delaying the sorted sequence corresponding to $\{x_4, x_5\}$ by B=3 cycles.

Multiple-median network:

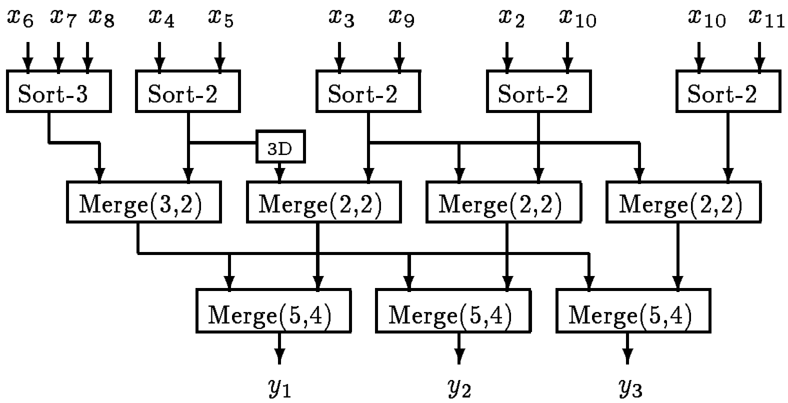


Figure 3: Block processing three outputs with window size 9 and block size 3. Here 3 outputs are computed every 3 cycles.

K	B	Number of comparators Ref[10]	Number of comparators Our Method
3	6	15	15
5	4	27	24
5	5	34	28
7	6	65	59
9	3	64	48

(a)

K=5 B	Number of comparators	Number of comp/output
1	8	8
2	12	6
3	17	5.7
4	24	6
5	28	5.6
6	34	5.8

(b)

Figure 4: (a) Comparison of the number of comparators of our method with those of [10] for various values of K and B. (b) Number of comparators/output for $K = 5$ and $B = 1$ through 6.

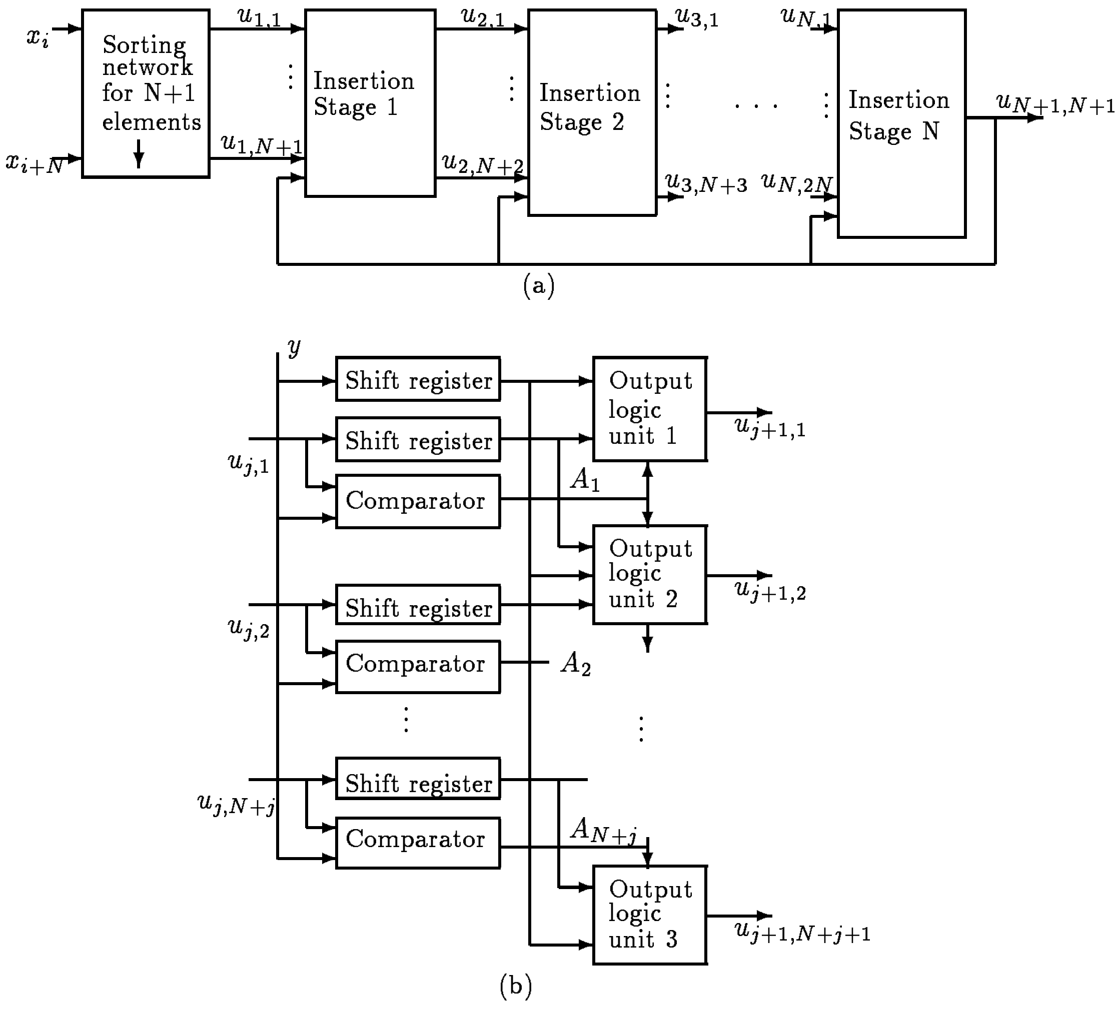


Figure 5: Sorting network based architecture for 1-D recursive median filtering. (a) Block diagram of the architecture (b) Implementation of insertion stage S_j.