

## An Efficient Pipelined VLSI Implementation of Rank Order Filter

Chun-Te Chen, Liang-Gee Chen, Tzi-Dar Chiueh, and Jue-Hsuan Hsiao

Department of Electrical Engineering  
National Taiwan University, Taipei, Taiwan, R.O.C.

### Abstract

In this paper, an efficient pipelined VLSI implementation of rank order filter based on the ordering property is proposed. In the previous works, it requires many iterations in bit-serial implementation of Positive Boolean Function[1] and have large number of counters, comparators and many inputs of AND-OR logic array in parallel architecture[2,3] to output  $M$ -th order, which is limited in real-time applications for large window size. Based on the ordering property of values of elements in  $m$ -array, we proposed a pipeline architecture to fold the parallel data flow in order to reduce hardware complexity without loss performance. The proposed design can also realize the concurrent search method for PBF to output  $M$ -th order sample. Compared with previous works, the hardware complexity is reduced from  $O(2^r)$  to  $O(2^{r/2})$  area-time complexity and the latency approaches to parallel implementation.

### 1 Introduction

Rank order filtering is a non-linear filtering technique which is used to preserve sharp edges in signal and remove impulsive noise efficiently. They are robust and well suitable for filtering data when the noise characteristic are not known. In recent years, there has been a strong tendency towards the applications of non-linear filtering to digital signal and image processing, for example, advance television systems. Several methods[1-10] have been presented to improve the efficiency of the algorithm on hardware implementation. A good survey of these architectures is found in [4].

Median filter is a special case of rank order filter. The existing architectures for these filters are either sorting algorithm or selection algorithm. The former one means the samples in every window must be sorted or its rank of every sample must be evaluated by linear array architecture or sort-network[4]. The latter one means the sample is first decomposed into a stack of binary signals. Then, the  $M$ -th larger value is found from this stack by positive boolean function[5,6,7].

Stack filters, introduced by Wendt *etal*[2], are a class of nonlinear filter based on a "stack" of binary processing circuits. In a stack filter, each of the  $r$ -bit binary-weight multilevel signals is decomposed into a  $2^r-1$  bit unit-weight signal. A "stack" of Positive Boolean Function (PBF's), which performs binary signal processing, is applied to each level of the unit-weighted data to create  $2^r-1$  unit-weighted binary data. The final output is obtained by converting the unit-weight signal to a binary-weighted signal. The area-time complexity for this stack filter is  $O(2^r)$ . Chen[1] proposed bit-serial tree-search method to reduce the hardware complexity to only one PBF. Gu[8] also proposed a recursive scheme to generate  $k$  output bits by an array of  $k$  processors. An alternative approach to realize a PBF is to use a counter and a comparator. We count the number of "1" in the input binary data and the output of the counter goes to a comparator. If it is greater than or equal to  $M$ , the output of the  $M$ -th rank-order will be "1", otherwise "0". This filter can be used as programmable rank-order filter, when the parameter  $M$  is variable. Based on this idea, N. Rama Murthy and M. N. S. Swamy [3] used many counters and comparators in highly parallel architecture to realized PBF. The area-time complexity for this stack filter is also  $O(2^r)$ . Recently, Lucke and Parhi [7] proposed a rank state machine using only one PBF unit for rank update logic circuit. The sample periods of [1] and [8] are bound by the latency of PBF units. However, if the window size is larger, direct realization of the PBF will be difficult. The main drawback of parallel implementation is that it requires large number of comparators, counters, and many inputs of AND-OR array.

In this paper, an efficient pipelined VLSI implementation of rank order filter based on ordering property is proposed. The conventional parallel implementation is folded into two pipelined stages based on the ordering property. The concurrent search method is also realized to output  $M$ -th order sample from stack efficiently. An alternative implementation is to fold the 1-D counters array in the parallel implementation[3] into 2-D counter array based on this ordering property. The wire connection of updating  $m$ -array is become simpler. Using the concurrent search method, the extraction circuit is also reduce. The deletion, insertion and extraction is executed concurrently in two

pipelined stages. The proposed design can be used as a basic building block for other form of nonlinear filter implementation.

## 2 An Efficient Pipelined Algorithm for Rank Order Filter

For one dimensional rank order filter, the running algorithm based on the threshold decomposition with m-array have been proposed [3,5,10]. There are two basic operations after each move of window. The first operation is to update some elements of m-array after the decomposition the insertion (deletion) sample value into binary weight signal. The second operation is to select or extract  $M$ -th order output from updated m-array. The  $k$ -th element of m-array is used to count the number of element of input window greater or equal to  $(k-1)$ . Thus, the values of element of m-array must be in descending order. And the step between every neighbor elements is same.

In other words, there is no new element will be created when new sample date is input. And any element of m-array will not be deleted when old sample data is deleted. Therefore, the shift operation to create or delete element is not required. We can fold the m-array to  $P$  segments to reduce updating circuit from  $O(2^q)$  to  $O(2^q)/P$  area-time complexity. The update procedure of each segment is pipelined independently. And the latency of updating m-array will be increased by  $P$  times.

There are only three different types of segment: up/down segment, restored segment, and mixed segment. In the up/down segment, each element is increased(decreased) by unity. The element in restored segment remains no change. There is only one mixed segment for each update cycle. And in the mixed segment, we need more computation to determine which element must be increment. Since the update procedure of up/down segment is uniform, these update data is stored in temporary registers until the insertion (deletion) segment occurred. To simply the proposed algorithm, the segments number  $P$  is set equal to  $2^q$ . Therefore, the first  $q$  bits (most significant bit) of input value is used to determine the types of the segments. The remain part of input value are used to determine which should be increased or not. The operation to determine segment type and accumulate in register is similar to construct of smaller m-array with scanning the first  $q$  bits (most significant bit) of sample in the window.

Obviously, this is two steps process. The first stage is used to determine the type of segment and store data. The second stage is to update the m-array of mixed segments. The latency for insertion or deletion can therefore be reduced from  $P$  cycles to 2 cycles.

At 1-D case, the extraction is performed after a pairs of deletion and insertion. In the parallel implementa-

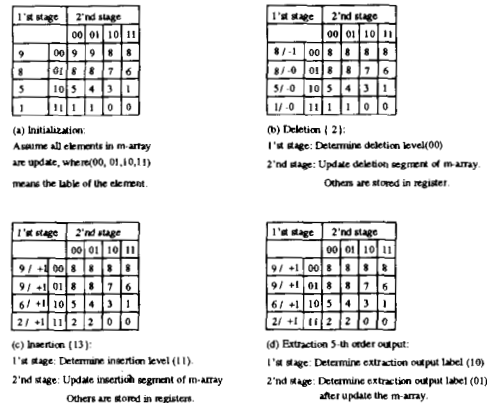


Figure 1: The procedure of deletion, insertion, and extraction are shown in (a), (b), (c), and (d), where the input stream 2,9,8,10,10,12,6,7,5,13,... for window size  $N=9$ .

tion, adders and comparators are used to evaluate the values of each element and the PBF unit is used to extract  $M$ -th order respectively. Here, the level of the m-array is labeled from  $0$  to  $2^q - 1$  in label registers. This label register is used to replace the PBF unit. Hence, we can also fold this operation into two pipeline stages as the same manner in insertion or deletion. The  $M$ -th order output is achieved by combining the first stage label output and the second stage label output. When the value of  $M$  is variable, this extraction procedure can be extended to output any order of sample.

To furtherly improve the performance, a concurrent search method is developed to output  $M$ -th order. The initial values of each element in m-array is equal to 0. The  $2^q$  complement of constant value  $M$  is added to the smaller m-array in the first stage. The carry out bits of these parallel row of adders are used to determine the extraction segment. The extract segment is occurred when the carry bits of segment is 1 and its descending neighbor segment is 0. Similarly, this extraction segment is used in the second stage to determine extraction element. The m-array data must be updated before extraction. Since there is no new data in extraction, the insertion(deletion) of new (old) pixel is executed immediately without waiting the extraction. The extraction operation is embed in the insertion(deletion) cycle. Therefore, the operation of second stage is reduced to only one pass of accumulation. Since there is no feedback loop, the operation of insertion, deletion and extraction are pipelined in this algorithm. The latency of the proposed algorithm can also reduce to half. It only requires 4 cycles to complete insertion, deletion and extraction operation for 1-D case.

Fig. 1 shows the executing sequences of our pro-

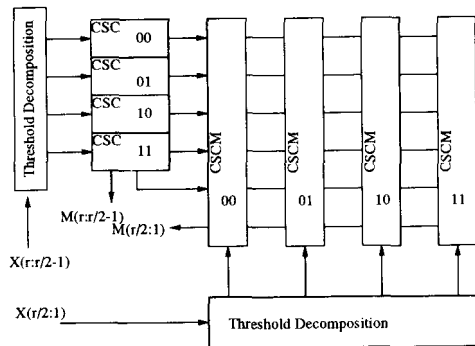


Figure 2: The block diagram of the efficient pipelined stage rank order filter, where X means input sample, M means the M-th order output, (00,01,10,11) means the label of segments or elements, q is set equal to r/2, and r=4 for this example.

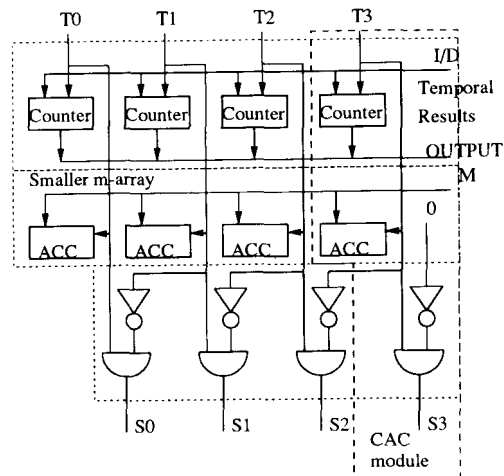
pose method, by consider the median filtering of the following input stream {2,9,8,10,10,12,6,7,5,13,...} with window size N=9.

### 3 Architecture for Efficient Pipelined Rank Order Filter

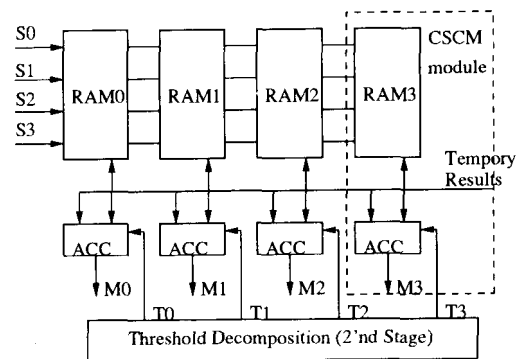
The block diagram of the proposed design is shown in Fig. 2. In the first stage, it contains one threshold circuit with q number of inputs and  $2^q$  number of concurrent search circuit (CSC) modules. In the second stage, it contains one threshold circuit with (r-q) number of input and  $2^{r-q}$  number of concurrent search circuit with RAM (CSCM) modules.

In the CSC module, it also contains one up/down counter to accumulated the temporal results and an accumulator to store segment result as shown in Fig. 3(a). The result in counter will be reset to 0 when it is sent to the second stage for m-array updating. The mixed segment for insertion or deletion is selected by the combinational logic operation of the threshold circuit output. Similarly, there is one accumulator and RAM in the CSCM module as shown in Fig. 3(b). The mixed segment for insertion, deletion or extraction in RAM is selected by the output of first stage. The detail of this mixed segment is updated by accumulator depending on the threshold output. When the updating of extraction segment is completed, it is ready to process the new RAM data. The comparison operation for extraction is realized by adder with checker bit. Hence, the parallel row of adders in Fig. 3(c) is used to extract M-th order output.

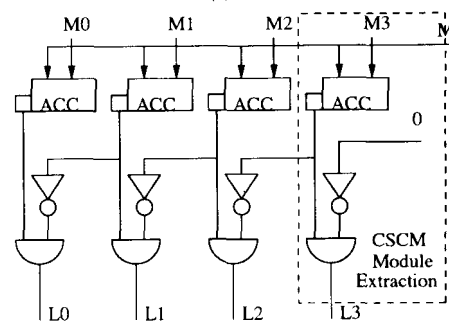
The q (most significant bit) bits of M-th order are determined at the first stage. At the same time the mixed segment for extraction is determined. Then, using this segment in the second stage, the remain part



(a)



(b)



(c)

Figure 3: The detail of block diagram of proposed design. (a) The block diagram of the CSC module. (b) The block diagram of the CSCM module. (c) The block of the concurrent search circuit.

Design Type	Parallel Design[3]	Proposed with RAM	Proposed with 2-D
Threshold	$2^r$	$2*2^{r/2}$	$2*2^{r/2}$
Counters	$2^r-1$	$2^{r/2}$	$2^r$
Comparator	$2^r$	$(3 \text{ or } 5)*2^{r/2}$	$2*2^{r/2}$
PBF	$2^r$	$2*2^{r/2}$	$2*2^{r/2}$
$T_d$	$4* \max(T)$	$4*T_{adder}$	$2*T_{adder}$
$T_p$	$\max(T)$	$T_{adder}$	$T_{adder}$

Table 1: The hardware complexity of the proposed design with parallel implementation in [3], where  $\max(T)$  means the  $\max(T_{pbf}, T_{adder})$ . The complexity of adder and comparator is equal.

of  $M$ -th is also determined after the adding operations.

The module design of CSC or CSCM aids in VLSI design. The extraction, insertion or deletions operation is performed by many CSC modules or CSCM modules in parallelism. In the first stage, the extraction procedure is easily modified and shared with the existing hardware. We need extra concurrent search circuit to embed the extraction cycle in the insertion cycle in the second stage. The operation of insertion, deletion and extraction can be pipelined in this architecture. To further improvement, three segments of RAM updating operation(insertion, deletion, and extraction) are concurrently executed by three adders in the second stage. The throughput rate approaches the parallel design[3].

An alternative implementation is used many counter in 2-D array to store the value of m-array. The threshold decomposition output of the first stage and the second stage is combined to determine which counter need count up or down for insertion or deletion concurrently. The proposed concurrent search circuit is adopted to output  $M$ -th order output. The benefit of this 2-D realization is the output load of threshold circuit is reduced. The wire of connection is also saved. The latency of this design is reduce to only two cycle time of adder. And the throughput rate is increased to  $1(\text{median output})/(\text{adder cycle})$  for 1-D case.

The comparison of the hardware complexity of the proposed design with parallel implementation in [3] is summary in the Table 1. Here, the delay time( $T_d$ ) and pipeline period ( $T_p$ ) are the evaluation criteria. The output of the parallel implementation is used PBF AND-OR array with  $2^r$  inputs. In the proposed design, we used  $2*2^{r/2}$  number of label registers.

#### 4 Conclusions

In this paper, an efficient pipelined VLSI implementation of rank order filter based on the ordering property is proposed. Based on the ordering property of values of elements in m-array, the parallel implementation is

decomposed into two pipeline stages to reduce hardware complexity without loss performance. To share the same hardware, we proposed a concurrent search method based on a parallel row of adders for extraction operation. In this way, the operation of insertion, deletion and extraction can be executed pipeline on same hardware. This configuration can support variations in filter size, rank order and word size. The latency of the proposed design is approach to highly parallel architecture. The area-time complexity is reduced to only  $O(2^{r/2})$ . The proposed design can also be used for 2-D median filter. The ordering property can be extend to reduce the number of AND and OR items in the PBF implementation.

#### References

- [1] K. Chen (1989). Bit-serial Realizations of a class on Nonlinear filter based on Positive Boolean Function *IEEE Trans. on Circuits and Systems*, vol. 36, 785-794,1989.
- [2] P.D.Wendt, E.J.Coyle, and N. C. Gallager (1986). Stack Filters *IEEE Trans. on Acoust.,Speech, and Signal Processing*, vol. ASSP-34, 898-911, 1986.
- [3] N.Rama Murthy and M.N.S.Swamy (1992). On the VLSI Implementation of Real-Time Order S-tatistic filters *IEEE Trans. on Signal Processing*, Vol. 40 1241-1252, 1992.
- [4] D. Richards (1990). VLSI Median filters *IEEE Trans. on Acoust.,Speech, and Signal Processing*, vol. ASSP-38, 145-153, 1990.
- [5] V.V.B. Rao and K.S.Rao (1986). A new algorithm for real-time median filtering *IEEE Trans. on Acoust.,Speech, and Signal Processing*, vol. ASSP-34, 1674-1675, Dec. 1986.
- [6] J.P.Kitch, E.J.Coyle, and N.C.Gallagher (1984). Median Filtering by Threshold Decomposition *IEEE Trans. on Acoust.,Speech, and Signal Processing*, vol. ASSP-32, 1183-1188, 1984.
- [7] J.P.Kitch (1987). Software and VLSI algorithms for Generalized Ranked Order Filtering *IEEE Trans. on Circuits and Systems*, vol. 34, 553-559, 1987.
- [8] Q.Gu and M.N.S.Swamy (1992). A Binary Logic Synthesis approach to the Bit-level Implementation of Generalized Rank-Order Filters *Proceedings of ISCAS 92*, 109-112.
- [9] L. E. Lucke and K. K. Parhi (1993). Block Processing for Rank Order Filtering using the Rank Order State Machine Architecture *Proceedings of ICASSP 93*, 1357-360.
- [10] E. Ataman, V.K.Aatre, and K.M. Wong (1980). A fast method for real time median filtering *IEEE Trans. on Acoust.,Speech, and Signal Processing*, vol. ASSP-28, 415-420, Aug. 1980.