High performance VLSI architecture for the modified SORT-N algorithm

Pavan Kumar Ganjimala[1], Subrahmanyam Mula[2]
Electrical engineering department, Indian Institute of Technology Palakkad, Palakkad, India
E-mail: [1]122014005@smail.iitpkd.ac.in, [2]svmula@iitpkd.ac.in

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937921

***Abstract*—Fast running sorting of streaming input data samples is very important in many applications such as order statistics, nonlinear filtering, MMax selective-tap adaptive filtering etc. This paper proposes a high performance VLSI architecture for the modified SORT-N algorithm for fast running sorting. Through analysis and also through synthesis results, we show that the critical path of the proposed architecture is almost independent of the sorting order N. ASIC synthesis results of the designed architecture shows that the proposed architecture has double the performance for N=1024 along with a reduction in area and power metrics compared to state-of-the-art architecture reported in literature and thus, it is potentially useful in real-time applications which have stringent throughput requirements.**

***Index Terms*—VLSI architectures, SORT-N, order statistics**

I. Introduction

Sorting of data is a common problem in many fields, each of them with different requirements and characteristics. There are several sorting algorithms and architectures available for batch processing [1]–[3], in which all the data to be sorted is available at the same time. However, for certain applications like median filtering, order statistics based filtering and partial update adaptive filtering, the input comes in a streaming fashion i.e., one sample at a time into the system. For these kind of applications, considering their real-time nature, dedicated ASIC/FPGA architectures are needed to support their high throughput requirements [4], [5]. We focus our architecture to the problem of continuous data stream sorting.

The problem of continuous data stream sorting has been addressed in many previous works in literature [6]–[8]. However, all of them assume a sliding window for sorting where the sorted data array of the window size. Each cycle a sample leaves the window and another sample enters the window creating a partially sorted array. However, in some applications such as linear detection of a weak signal in the presence of impulsive noise [9], the last N samples streamed are sorted and utilized to make a decision. Once a decision is made, the sorted register array is reset to accept N new samples. Therefore existing methods to sort streaming data are not suitable in the mentioned application, since the entire sorted array is reset before the next sort. Hence, in [10], the authors devised a method and it's VLSI architecture, called "SORT-N", to perform running sort on N incoming samples in N clock cycles. However, as we show the critical path of the architecture proposed in [10] increases logarithmically with N which can become a potential bottleneck for large N. In this work, we address this issue. We first review the architecture presented in [10].

II. Background

The block diagram of the SORT-N architecture that was proposed in [10] is shown in Fig. 1. The design sorts N samples in N clock cycles in the ascending order. The architecture has N registers and each register has a corresponding comparator (CMP), multiplexer (MUX) and a multiplexer controller (MUX_CTRL). A leading-one-detector (LOD) and a counter are used to decode the register to which the input sample has to be loaded. Each incoming sample arriving every clock cycle is stored in REG_{IN} and is compared with all the samples in the N registers in parallel through comparators CMP_0, CMP_1, ... $\text{CMP}_{\text{N-1}}$ to produce a comparator code $cmp[0:N-1]$. The comparator produces a logical high ('1') if the input sample is lesser than the corresponding register value and a logical low ('0') if input sample is greater. The comparator code generated is then fed to the LOD. Given a binary code, an LOD finds the position of the leading one in the code. Therefore the LOD block's output indicates the register position where the incoming sample has to be inserted in the register array. The structure of the LOD block and it's effect on critical path delay are discussed in the next section. Before inserting the sample, the values in that register and to it's right need to be shifted right. The MUX_CTRL logic generates the select signal to the corresponding MUX to select the appropriate input by comparing each register's position with the LOD generated position. Each MUX has three inputs- previous register input (shift operation), present register feedback (retain operation) or the incoming sample (load input operation) selected based on MUX_CTRL logic.

In SORT-N an LOD has been used to decode the comparator code and find the position where the new sample has to be inserted. It can be observed that the register array at any point of time is always sorted (excluding the reset valued registers) and thus the comparator code produced is always a series of 0's followed by 1's. We analyse in the next section that for such a comparator code pattern an LOD is not necessary and can be replaced with a simpler design. Additionally in a special scenario where the input sample is greater than all the samples a comparator code with all zeros is generated for which the LOD may not produce a valid output. In order to counter this problem, SORT-N uses an N bit counter to generate the load position. The index generator block (IG) has been used

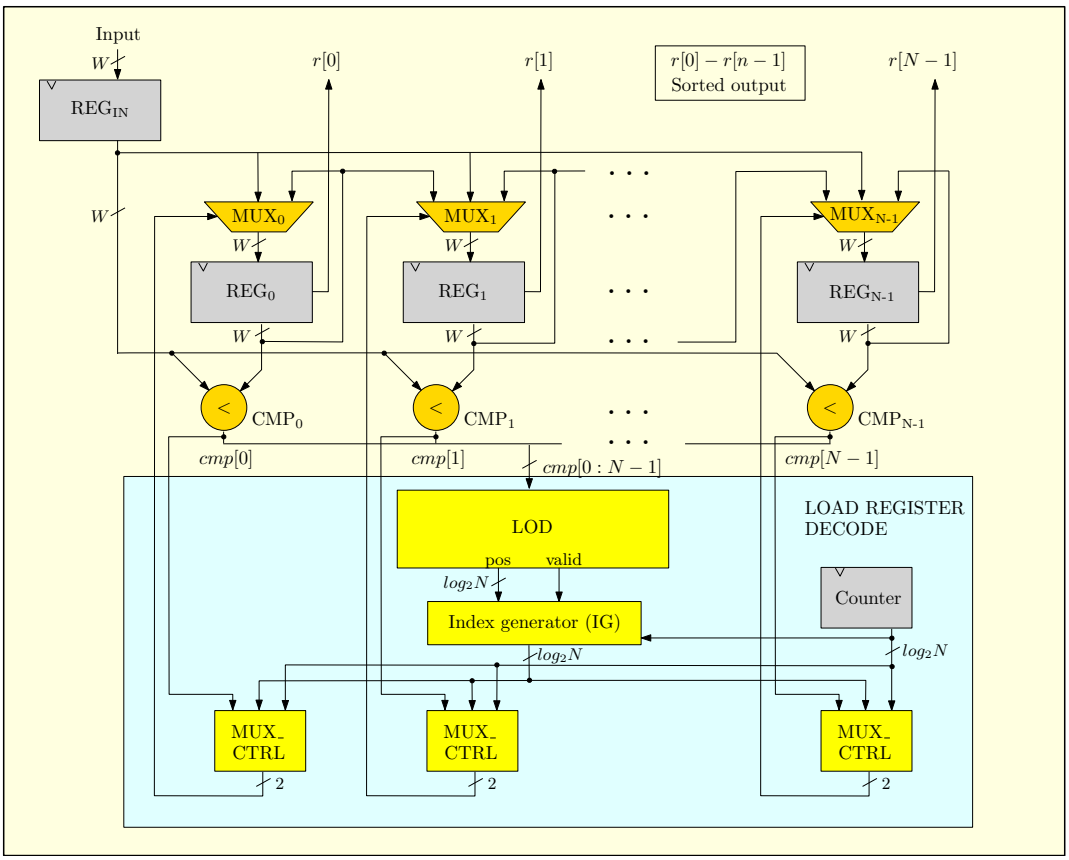

Fig. 1: SORT-N Architecture [10]

to select between the LOD and counter value based on LOD valid signal. We show in the next section that by sorting in the descending order, the counter circuitry can be avoided.

III. Proposed XSORT-N Architecture

The block diagram of the proposed VLSI architecture coined "XSORT-N" is shown in Fig. 2. Similar to SORT-N, the proposed architecture performs the sorting of N samples in N clock cycles and has a set of N registers and corresponding multiplexer (MUX) and comparator (CMP). Unlike the previous approach this architecture performs sorting in the descending order. Initially all the N registers are reset to 0 and the input sample is stored in register REG_{IN} every clock cycle. The input sample stored in REG_{IN} is compared with all the samples in the N registers in parallel through comparators CMP_0, CMP_1, ... $\text{CMP}_{\text{N-1}}$. Since the sorting is done in descending order in this architecture, when the input sample is greater than the sample stored in a register, the corresponding comparator produces a logical high ('1') or else a logical low ('0'). It should be noted that comparator CMP_0 to $\text{CMP}_{\text{N-1}}$ outputs are always a stream of 0's followed by 1's since the register values are always sorted in descending order at any point of time. For example if the registers REG_0-$\text{REG}_{\text{N-1}}$ contain $8, 6, 5, 3, 2, 0 \ldots 0$. For an input 4, the first three values have to be retained, the input has to be inserted into the 4^{th} place and the subsequent values right shifted. The comparator code that would be produced is $0, 0, 0, 1, 1, 1, \ldots 1$. It can be observed that the load register can be decoded by just a simple XOR operation on neighbouring comparator outputs ($xor[x] = cmp[x-1] \oplus cmp[x]$). The code generated after XOR operation is $0, 0, 1, 0, 0, \ldots 0$, where the position of 1 corresponds to REG_3. It must be noted that there is no XOR output corresponding to REG_0. So the place where the comparator code transitions from 0 to 1 is the place where the XOR gate output would be high and the corresponding MUX would switch to the load operation. The rest of the registers to the right of the load register will have an XOR output 0 and CMP output 1 and will be right shifted to accommodate the new sample. Whenever a comparator code is 0, the corresponding register value is always retained and does not depend on XOR outputs. The truth table for MUX_1 - $\text{MUX}_{\text{N-1}}$ is given in Table I. An XOR output of 1 and a corresponding CMP output of 0 can only occur at a 1 to 0 transition in the comparator code. Since the numbers are always sorted in descending order the only valid code is a series of 0's followed by 1's, therefore such an XOR-CMP combination is invalid. At the end of the sorting procedure, the N sorted samples are stored in registers REG_0 - $\text{REG}_{\text{N-1}}$ in the descending order and are available as parallel outputs $r[0] - r[N-1]$. After the sorted samples are utilised, the

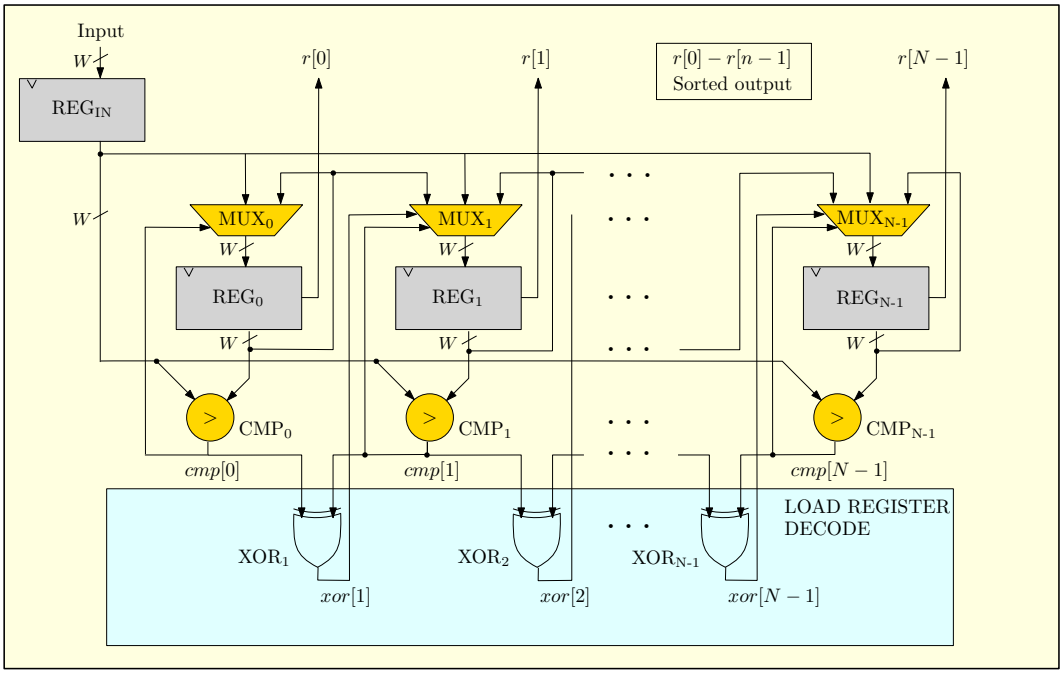

Fig. 2: Proposed XSORT-N Architecture

registers are then reset to accept another round of N samples.

The truth table of multiplexer MUX_0 which loads input to REG_0 is shown in Table II. Whenever $cmp[0]$ is 0, i.e input sample is smaller than REG_0 value, then REG_0 value is retained as in the other multiplexers. When $cmp[0]$ is 1, the input sample is greater than REG_0 value and since at any point of time, the largest value is stored in REG_0, the entire register array is shifted right and input sample is directly loaded to REG_0. This scenario where the value of $cmp[0]$ becomes 1 occurs when the input sample is greater than all the register values. As can be seen there is no need of a counter as was used in the original SORT-N design. Another corner scenario is when the input sample is zero, which would produce a comparator code containing all 0's, which sets all the registers in the retain mode of operation. Since the reset value of all registers is 0, the input sample need not be loaded anywhere. After the N samples are input in N clock cycles, if there are m zero samples, then the last m registers would be containing the zeros.

TABLE I: Truth table for MUX_x, $x = 1, 2 \ldots N-1$

$xor[x]$	$cmp[x]$	Operation	Output
0	0	Retain	REG_x
0	1	Shift	REG_{x-1}
1	1	Load	REG_{IN}
1	0	Invalid	–

TABLE II: Truth table for MUX_0

$cmp[0]$	Operation	Output
0	Retain	REG_0
1	Load	REG_{IN}

A. Critical path analysis

To analyze the timing of SORT-N and it's maximum operating frequency, the critical path of the design has to be identified. In the original SORT-N design the critical path is the path consisting of the multiplexer select logic. It is given as $T_{cp1} = T_{CMP} + T_{LOD} + T_{IG} + T_{MUX\_CTRL}$, where the delays T_{CMP}, T_{LOD}, T_{IG} and T_{MUX_CTRL} are the delays of the CMP, LOD, IG and MUX_CTRL blocks. A major contributor to delay is the LOD block. The diagram of a typical tree structured LOD block [11] is shown in Fig. 3. It can be seen that as the LOD input sample width doubles, one new layer of LOD decoding is required and therefore LOD delay increases linearly as input bit width doubles. Therefore the critical path delay of SORT-N for a fixed sample width and sorting order N is of $O(log_2 N)$, due to the LOD.

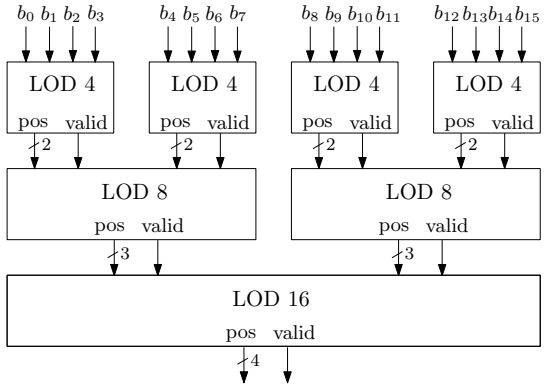

Fig. 3: A typical LOD tree structure

TABLE III: Synthesis Results in GSCLIB045 45nm CMOS.

Design	N	Min Delay (ns)	Max CF (GHz)	Power (mW)	Area (μm^2)	PDP (fJ)	EDP $(mW \cdot ns^2)$
SORT-N [10]	32	0.720	1.38	27.43	21015	19.75	14.22
	128	0.990	1.01	91.49	87063	90.57	89.67
	512	1.210	0.83	333.93	376478	404.058	488.91
Proposed XSORT-N	32	0.481	2.08	39.02	22514	18.77	9.03
	128	0.525	1.91	138.58	83002	72.76	38.20
	512	0.565	1.77	512.56	301999	289.59	163.62

CF: Clock frequency, PDP: Power delay product, EDP: Energy delay product

In the proposed XSORT-N design, similar to the original design the multiplexer select logic path is the critical path. From Fig. 2 it can be seen that the critical path delay is given by T_{cp2} = T_{CMP} + T_{XOR} + T_{MUX}, where T_{CMP}, T_{XOR} and T_{MUX} are the delays of the CMP, XOR and the MUX blocks. Comparing the original SORT-N's critical path, it can be seen that the LOD delay which increased logarithmically with N has been replaced by a fixed delay XOR gate. None of the blocks in the critical path have a dependancy on the sorting order N. Therefore it can be seen that the proposed XSORT-N design has a lower critical path delay and in addition to that, delay remains constant even as the sorting order N increases. Significant delay reduction could be obtained by the proposed design over SORT-N, especially for high values of N.

In both the designs, ideally every register to register path has the same critical path as described above, but practically speaking, the path from REG_{IN} to any other register would have more delay because of the high fanout of REG_{IN}. The high fanout problem is common to both the designs and increases as the value of N increases. It can be mitigated by the use of various fanout optimization techniques like usage of buffers, etc.

IV. VLSI Implementation Results

The proposed XSORT-N architecture as well as SORT-N have been implemented in Verilog HDL and simulated using Cadence NCsim simulator to verify the sorting process. SORT-N contains a control path consisting of a counter to indicate that sorting is ready once N samples are read. For a more fair comparison a $log_2 N$ bit counter was also added to XSORT-N to generate the ready control signal. Both the designs have been synthesized using Cadence Genus synthesis tool with Cadence's GSCLIB045 45nm library. The synthesis results of the two designs for a sample bit width $W = 16$ and for sorting order $N = 32, 128,$ and 512 are shown in Table III.

From Table III, it can be observed that the XSORT-N design has a higher maximum attainable clock frequency (CF) and lesser critical path delay compared to the SORT-N design for all N values. As the N value increases the critical path delay also increases more in the case of the SORT-N design because of the LOD block. It can also be seen that the area is also slightly lesser for XSORT-N especially as N increases. The power dissipation of XSORT-N is higher than SORT-N, but since both are synthesized at different target frequencies, power metrics cannot be compared directly. To remove the clock frequency aspect in power we take the power delay product (PDP) and energy delay product (EDP) as the metrics [12] which are more meaningful in this scenario. It can be seen that both PDP and EDP are lower for XSORT-N. Therefore the increase in power reported in XSORT-N is due to it's higher operating frequency. XSORT-N will have lower power when both are synthesized at the same target frequency. The XSORT-N design has around $36.5\%, 57.4\%$ and 66.5% reduction in EDP over SORT-N for $N = 32, 128, 512$ respectively.

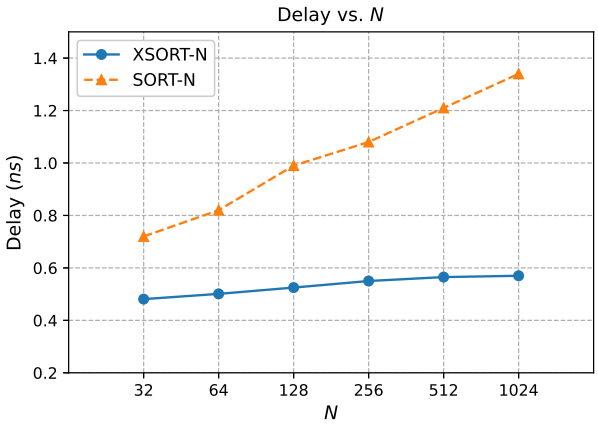

Fig. 4: Delay vs N plot

The change in the critical path delay as the sorting order N is doubled is shown in Fig. 4. It can be seen clearly that in the case of SORT-N, the delay increases linearly as N doubles, whereas with the proposed XSORT-N the delay remains more or less constant. The slight increase in delay observed for XSORT-N with N is because of the increase in fanout of REG_{IN}.

V. Conclusions

In this paper, a high performance version of the SORT-N design coined the XSORT-N which performs continuous sorting of N streaming samples in N clock cycles was presented. The LOD block which was the main timing bottleneck in the SORT-N design was replaced by a novel XOR gate based design. By analysis and through synthesis it was shown that the proposed design achieved less critical path delay and is independent of the sorting order N. For high values of N, the proposed design achieves very high performance along with lesser area and power metrics and could be very useful in real time applications having high throughput requirements.

ACKNOWLEDGEMENT

This work was supported in part by Prime Minister's Research Fellowship (PMRF), Ministry of Education (MoE), GoI (to Pavan Kumar Ganjimala) and in part by "IIT-Palakkad Research Seed-Grant for Faculty Members"

REFERENCES

[1] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte and K. Compton, "Modular Design of High-Throughput, Low-Latency Sorting Units," in *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1389-1402, July 2013.

[2] A. Norollah, D. Derafshi, H. Beitollahi and M. Fazeli, "RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1601-1613, July 2019.

[3] W. -T. Chen, R. -D. Chen, P. -Y. Chen and Y. -C. Hsiao, "High-Performance Bidirectional Architecture for the Quasi-Comparison-Free Sorting Algorithm," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1493-1506, April 2021.

[4] G. M. Blair, "Low cost sorting circuit for VLSI," in *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, no. 6, pp. 515-516, June 1996.

[5] D. Koch and J. Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-Time Reconfiguration on FPGAs for Large Problem Sorting", *Proc. Symp. Field Programmable Gate Arrays*, pp. 45-54, 2011.

[6] I. Pitas, "Fast algorithms for running ordering and max/min calculation," in *IEEE Transactions on Circuits and Systems*, vol. 36, no. 6, pp. 795-804, June 1989.

[7] A. A. Colavita, A. Cicuttin, F. Fratnik and G. Capello, "SORTCHIP: a VLSI implementation of a hardware algorithm for continuous data sorting," in *IEEE Journal of Solid-State Circuits*, vol. 38, no. 6, pp. 1076-1079, June 2003.

[8] S. Lin, P. Chen and C. Lin, "Hardware Design of an Energy-Efficient High-Throughput Median Filter," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65, no. 11, pp. 1728-1732, Nov. 2018.

[9] S. R. K. Vadali, P. Ray, S. Mula and P. K. Varshney, "Linear Detection of a Weak Signal in Additive Cauchy Noise," in *IEEE Transactions on Communications*, vol. 65, no. 3, pp. 1061-1076, March 2017.

[10] S. R. K. Vadali, S. Mula, P. Ray and S. Chakrabarti, "Area Efficient VLSI Architectures for Weak Signal Detection in Additive Generalized Cauchy Noise", in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 6, pp. 1962-1975, June 2020.

[11] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 1, pp. 124-128, March 1994.

[12] P. K. Meher and S. Y. Park, "Area-Delay-Power Efficient Fixed-Point LMS Adaptive Filter With Low Adaptation-Delay," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 2, pp. 362-371, Feb. 2014.