FPGA IMPLEMENTATION OF SORTING ALGORITHMS

Magesh.V[1], Megavarnan.S[2], Pragadish.A[3], Saravanan.S[4]
[1]Assistant Professor, [2,3,4]U.G. Scholar,
Department of ECE, Velammal Engineering College,

Abstract: As sorting is one of the most fundamental concepts. An efficient sorting network as to be designed which need to be feasible. This paper focuses on the architecture which sorts values much faster such that the delay is reduced. In this paper, the sorting networks are functionally verified using Verilog HDL. All the network models which are discussed in the paper are simulated with Xilinx ISE. The sorting networks are designed, synthesized, timing summary is analyzed and their RTL diagrams are examined. Based on the results , the comparison is made between the existing sorting network designs and the proposed network design. The experimental results show that the delay is reduced and the speed is increased in the proposed sorting network design.
Keywords: Sorting, area, speed, delay, FPGA implementation, VLSI.

I. INTRODUCTION

1.1 Introduction to Sorting

Sorting is, without doubt, the most fundamental algorithmic problem that was faced in the early days of computing. In fact, most of the computer science research was centered on finding the best way to sort a set of data. There is probably a good reason to make sorting that important.
Supposedly, 25% of all CPU cycles are spent sorting

- Sorting is fundamental to most other algorithmic problems, for example binary search.
- Many different approaches lead to useful sorting algorithms, and these ideas can be used to solve many other problems.

1.2 The Process of Sorting

Given a data set $\{x_1, x_2, \ldots x_n\}$ we need to find a permutation such that the set is sorted in increasing or decreasing order. However, if we look for all permutations of $\{x_1, x_2, \ldots x_n\}$, then there are n! of them around. Needless to say, n! is huge even for a small n such as n=20. However, if you can find all permutations of the data set, then we can determine if any of those lists are sorted in O(n) time. Therefore we need to think of sorting a list as a different activity other than finding a permutation that is sorted. Let us make some definitions first.

A pair of elements is inverted if $x_i > x_j$ for $i < j$. Therefore a non-sorted list has at least one inverted pair. Now we can define the act of sorting as removing all inverted pairs in the list. In other words, if you can prove that the number of inverted pairs in a list of elements is zero, then the list is sorted. Hence our goal is to study algorithms that remove inverted pairs from a list of elements.

1.3 Issues in Sorting

There are many issues that need to be considered when sorting a list. We need to consider whether we need to sort the list in increasing or decreasing order. Clearly we can use the same algorithm in both cases. All we need to do is to change the comparison criteria from > to < or vice versa. What about equal keys? Do we change their order or leave them wherever they are? How about non-numerical keys such as Strings? How do we sort them? What if we want to sort a list of names by two criteria's? First by the last name, then by the first name?
There is one thing that we assume for any list that needs to be sorted. We assume that keys in the list can be “compared” by some criteria.

1.4 Applications of Sorting

There are many applications of sorting. Once a list is sorted many questions about the list can be answered easily. We can efficiently find an element in a sorted list using Binary Search. Binary search requires only O(log n) operations in finding an element. We can also determine in O(n) if a sorted list has duplicates. We can construct a frequency distribution of the list if the list is sorted, or find the median and mode of the list in O(1) and O(n) respectively. We can find the kth largest element in a list in O(1) time.The sorting of a series of numbers is a very important task, which embraces many different applications, from banking , signal processing techniques, such as order statistics, non-linear filtering to communication switching systems to image processing or pattern recognition techniques . In this paper the VHDL design of an elementary sorting unit is presented. The main contribution of this paper is to describe a case study of a simple and general approach to VLSI sorting device. Both ascending and descending ordering can be implemented in the proposed architecture.

1.5 Sorting technology in hardware

In hardware architecture, a sorting module is generally composed of a series of the compare-swap unit. Two values are compared using a comparator and the result is used to control the two multiplexers that select certain values to generate outputs in an increasing or decreasing order. Two different symbols ⊕and⊖ are adopted to represent two kinds of units. It is possible to create hardware implementations of existing software algorithms, with or without parallelization of the problem. The running time of the comparison-based software sorting algorithm is asymptotically limited by the lower bound Ω(nlog2n). However, the corresponding hardware design may have better performance. Several hardware sorting algorithms will be analyzed in this section.

II. EXISTING AND PROPOSED SYSTEM

2.1 Existing system

Here we discuss the most commonly used sorting units and their designs. The basic and the most commonly used sorting methods are the bubble sort, selection sort, odd-even merge sort, bitonic merge sort. Among which bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. Bitonic Sort is a classic parallel algorithm for sorting.

- Bitonic sort does O(n Log 2n) comparisons.
- The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [does O(nLogn) comparisons], but Bitonic sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data. Therefore it is suitable for implementation in hardware and parallel processor array.

Structure of the comparator:

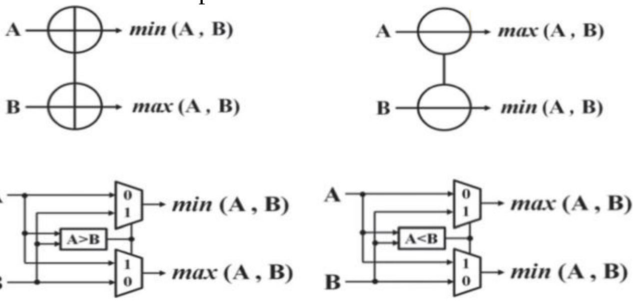


Fig 1.The increasing (a) and decreasing (b) comparing block. (c) and (d) are the detail architectures.

This fig 1. shows the types of comparators used in the design , one comparator compares and arranges the elements in ascending order and other comparator arranges the elements in descending order.

2.1.1Bubble Sort

For small inputs, bubble sort is a feasible solution and is also easy to implement. In each round, the largest (or smallest) sample is selected by a series of comparisons. The algorithm requires M comparison and switching events in the first round when the input size is M, M-1 events in the second round, M-2 events in the third round and so on until the complete sorted result is generated. The running time is O(n log n)where M is the input size. Parallelization is a well-known solution to enhance the performance.

Example:
First Pass:
(5 1 4 2 8) –> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 5 4 2 8) –> (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) –> (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) –> (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
Second Pass:
(1 4 2 5 8) –> (1 4 2 5 8)
(1 4 2 5 8) –> (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) –> (1 2 4 5 8)
(1 2 4 5 8) –> (1 2 4 5 8)

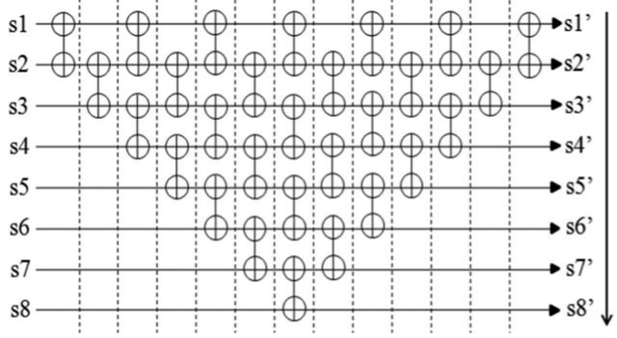


Fig 2.parallel architecture of eight input bubble sort

Fig 2. shows a hardware design for parallel bubble sort, where comparison blocks in the same column are executed in parallel. Although the total number of comparing units is as same as that in the sequential version, a few operations could be executed simultaneously in a certain pipeline stage. The overall pipeline stage is defined as 2M-3, and the runtime can be reduced to O(M).

2.1.2 Batcher's Odd-Even Merge Sort

The odd-even merging unit proposed by Batcher merges two sorted sequences into a complete sorted result. A sorting network can be recursively constructed using the merging unit. An M-input odd-even merging unit is denoted by OE-M, where M should be the power of two. The sorted sequence could be generated through a series of parallel merging units from OE-2s, OE-4s, OE-8s ... to OE-M. The architecture is parallel and feasible for pipeline design. To sort a data set with 2^P samples, there are 2^P-1 OE-2s in the first stage, 2^P-2 OE-4s in the second stage, and soon, until there is one OE-2^P in the final stage. Furthermore, an OE-2^P merging unit could be subdivided into P stages. The time complexity of an odd-even merging network with M inputs can be represented by O($\log 2^2$ M) because there are 1 + 2 +…..+ $\log_2$ M stages in total, and the area complexity is O(M ✖ $\log 2^2$ M). Fig. 3 illustrates an example of an eight-input odd-even merge sorting network composed of four parallel OE-2s, two parallel OE-4s, and one OE-8. The pipeline levels are 6 and there are 19 increasing comparison blocks.

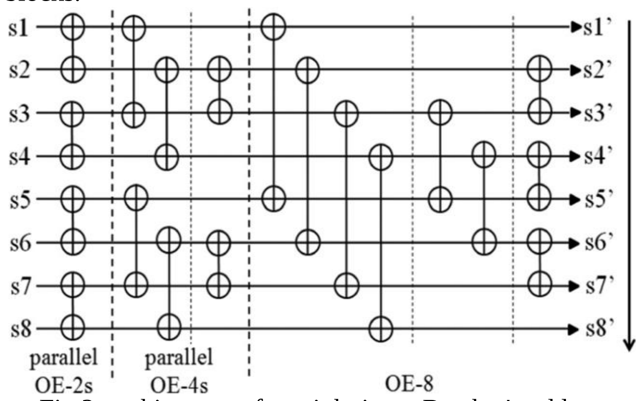


Fig 3. architecture of an eight input Batcher's odd-even merge sort

Fig 3. illustrates an example of an eight-input odd-even merge sorting network composed of four parallel OE-2s, two parallel OE-4s, and one OE-8. The pipeline levels are 6 and there are 19 increasing comparison blocks.

2.1.3 Bitonic Merge Sort :

Bitonic sort is another sorting network proposed by Batcher . A bitonic sequence is composed of one sequence in increasing order and another one in decreasing order. The bitonic merging unit merges the two sequences with equal length into a complete sorted result. Bitonic sort has been used widely because of its regular structure, which makes it considerably simpler to implement than an odd-even sorting network. Similarly, the input size of bitonic merging unit should be a power of two. The M-input merging unit receives an ascending and a descending sequence, and both of them contain M/2 samples. It is called a BM-M, where M can be represented as 2^P. To construct a complete 2^P-input bitonic sorting network, a series of bitonic merging units are applied recursively to generate the bitonic subsequence. The 2^P-input bitonic sorting network consists 2^P-1 parallel BM-2s in the first level, 2^P-2 parallel BM-4s in the second level ... and one BM-2^P in the final level. The time complexity and area cost are the same as those of the odd-even sorting network.

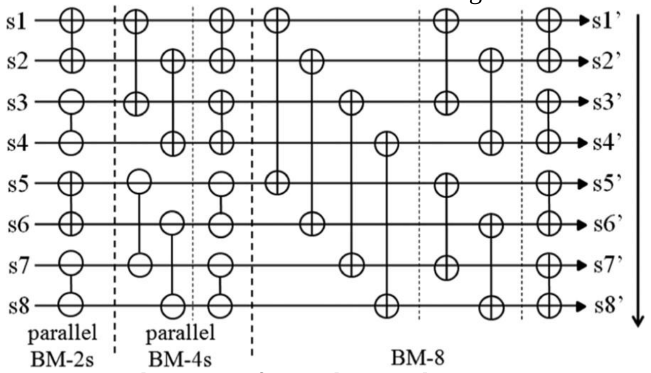


Fig 4.architecture of an eight input bitonic merge sort

Fig 4. illustrates an example of an eight-input bitonic sorting network composed of four parallel BM-2s, two parallel BM-4s, and one BM-8. The pipeline levels are 6, and there are 24 comparison blocks. A few of the comparing blocks produce increasing sequences, whereas others produce decreasing results, which is the most notable difference between the odd-even and the bitonic sorting networks.

2.2 Proposed system

The existing system is non-pipelined, so the latency and delay will be same for it. But in our proposed system, since we are introducing the pipeline concept the latency and delay will differ. The Latency is the delay from input into a system to desired outcome; the term is understood slightly differently in various contexts and latency issues also vary from one system to another. Here in our proposed system, since we are dividing the execution stages into three segments by using the registers, the delay will be considered as the maximum time required by anyone of the combinational block to get the data inside and to send the data signals outside the logic block. This is explained as follows

For a non-pipelined combinational circuit, the block diagram is as follows

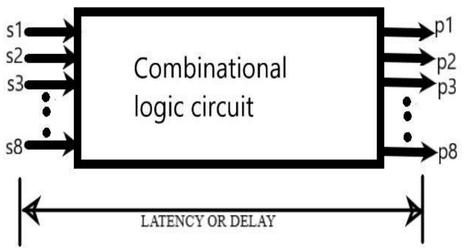


Fig 5. Eight I/O combinational logic circuit without pipeline stages

The eight input/output combinational logic circuit without pipeline stages is shown in the fig 5. For a combinational circuit without pipeline stages the latency and delay are both same.

Combinational circuit with pipelined stages

The combinational circuit is now pipelined by making use of the registers, such that the entire combinational logic circuit will be divided into three combinational logic blocks. This is explained as follows in the fig 5.

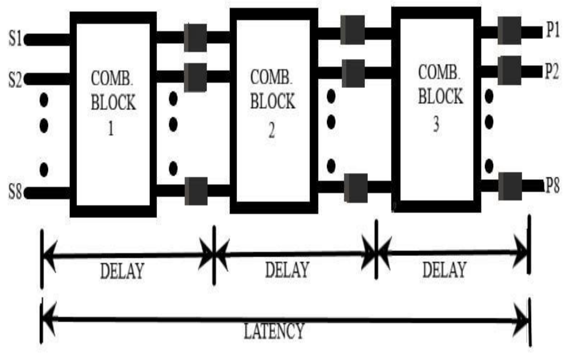


Fig 6. Eight I/O combinational logic circuit with pipelined stages

The fig 6. shows the combinational logic circuit with pipelined stages. In this circuit we are separating the combinational block into three segments by using the registers in-between each comparison stage, by doing so we are reducing the delay for each stage.

The delay will be given as,

delay = max (d1,d2,d3)

where d1, d2, d3 are the delay corresponding to each combinational blocks 1,2,3 respectively .

we are taking the maximum delay out of the three delay values, the reason for this is to avoid overloading of the buffer storage.

2.2.1 Pipelined Bitonic merge sort

Therefore to reducing the delay , we have used the pipeline concept in our proposed system. The design for our system is as follows,

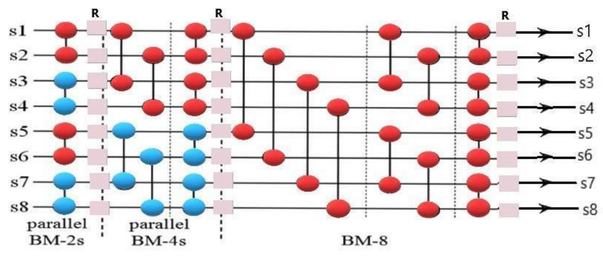


Fig 7. The architecture of an eight-input pipelined bitonic merge sorting network .

The architecture of the pipelined bitonic sorting network is shown in the fig 7. In our proposed system, we are applying the pipeline concept to the existing bitonic merge system. For making the proposed system we are providing adequate buffering storage between the pipeline stages , and this is done by making use of registers. As we are use pipeline in the circuit, which leads to increase in latency but our main aim is to decrease the delay. In this proposed system the registers are placed at respective positions at the pipelined stages.

III. 3. RESULTS AND DISCUSSION

3.1 General

VERILOG HDL is a hardware description language (HDL). A hardware description language is a language used to describe a digital system, for example, a computer or a component of a computer. One may describe a digital system at several levels. For example, an HDL might describe the layout of the wires, resistors, and transistors on an integrated circuits (IC) chip, i.e., the gate level. An even high level describes the register and the transfer of vector of information between the registers. This is called as Register Transfer Level (RTL). VERILOG supports at all these levels. However, this handout focuses on only the portion of VERILOG support the RTL level.

We have synthesized , implemented our proposed sorting network design in Xilinx ISE 9.2 , and simulated the design in ModelSim ALTERA 6.5b.

3.2 pipelined bitonic merge sort

The pipelined bitonic merge sort which we designed has been synthesized and implemented in Xilinx ISE design suite. The RTL view is produced using the schematic viewer which is a tool provided by Xilinx ISE . And the simulation is carried out using ModelSim.

The obtained results will be as follows,

3.2.1 pin configuration

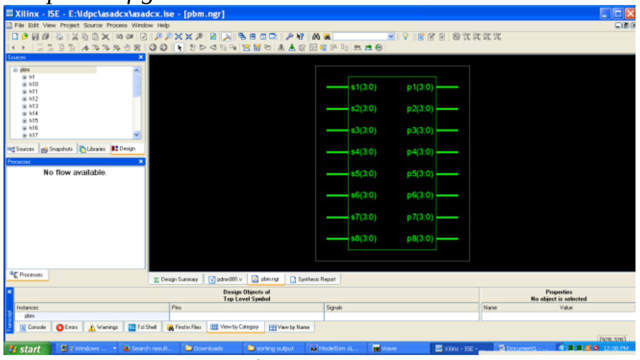


Fig 8. Pin diagram for pipelined bitonic merge sort

The fig 8. shows the pin diagram of pipelined bitonic merge sort network, in which the s1,s2,...,s8 are the input pins and p1,p2,...,p8 are the output pins. This pin diagram enable you to view the list of input and output ports available in the design.

3.2.2 RTL schematic

This schematic is generated after the HDL synthesis phase of the synthesis process. It shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, that are independent of the targeted Xilinx device. RTL View is a Register Transfer Level graphical representation of your design. This representation (.ngr file produced by Xilinx Synthesis Technology (XST)) is generated by the synthesis tool at earlier stages of a synthesis process when technology mapping is not yet completed. The goal of this view is to be as close as possible to the original HDL code. In the RTL view, the design is represented in terms of macroblocks, such as adders, multipliers, and registers.

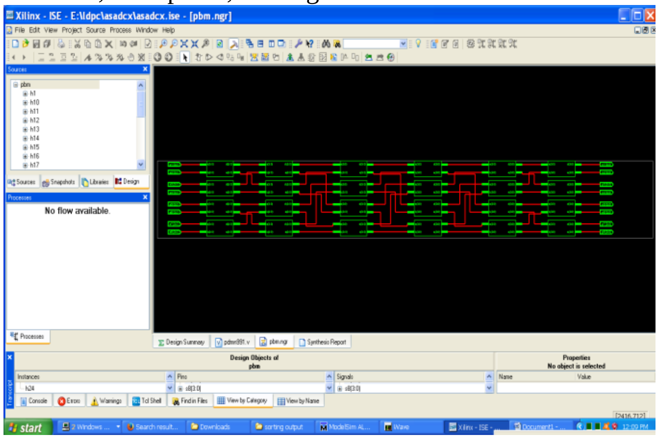
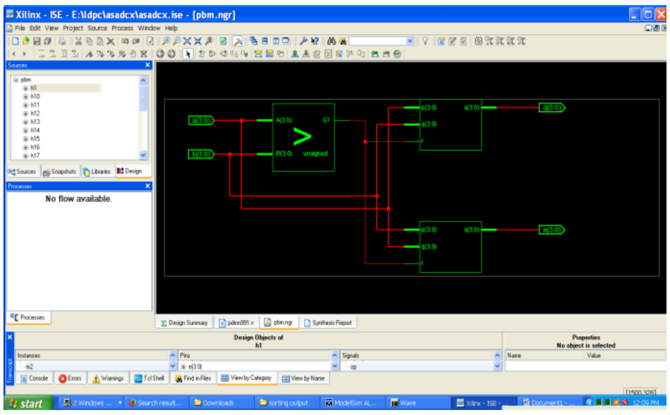


Fig 9. RTL view for pipelined bitonic merge sort

The fig 9. shows the RTL view for the Pipelined bitonic merge sort network design. This RTL view is produced after the synthesis process. From the obtained RTL view, we can able to view the design flow of the digital signal between hardware registers and also the logical operations performed on those signals. Through this RTL view, we can able to analyse our design from various perspectives.

Expanding blocks

By clicking on the blocks in the RTL view, we can able to see the detailed view of the internal configuration in each of the blocks.



(a)

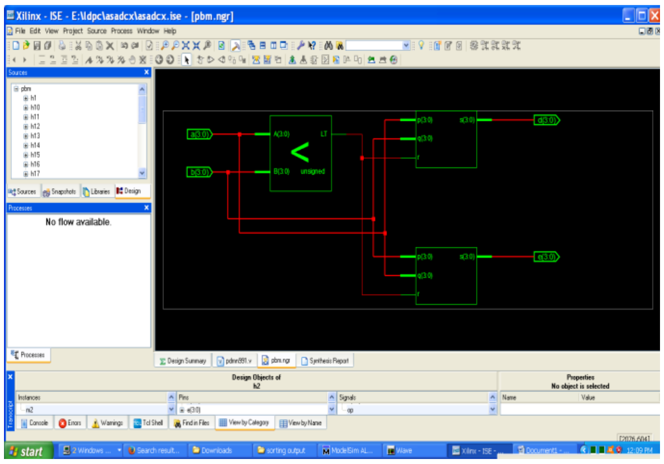

(b)

Fig 10 (a) and (b) are the comparator block diagrams

The fig 10. (a) and (b) shows the comparator blocks , where the fig(a) comparator block is intended to do the comparison in ascending order and the fig(b) comparator block is intended to do the comparison in descending order.

3.2.3 simulation results

We are using ModelSim Altera 6.5b starter edition to simulate our Verilog code, to determine our thinking is right. By simulation we can take into account the time delay . From the simulation we can able to see how the input signals move, get compared by the comparators and then till generation of the final sorted output signals.

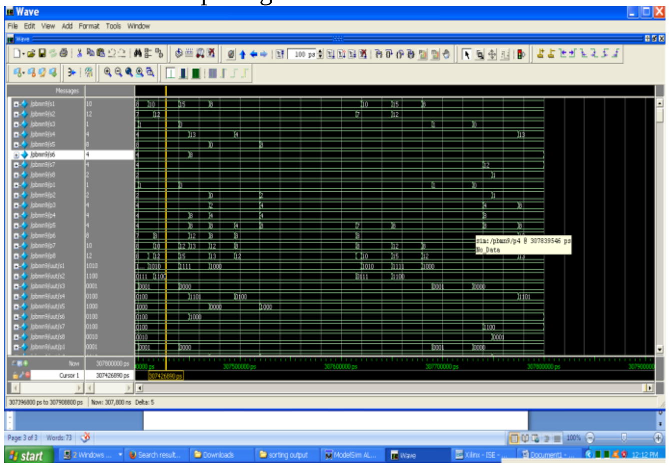


Fig 11. Wave diagram for eight-input pipelined bitonic merge sort

The fig 11. shows the wave diagram for eight input pipelined bitonic merge sort, this is generated by using ModelSim.

3.2.4 AREA AND DELAY COMPARISON

Table 1. Comparison between the existing system and proposed system in terms of area and delay

Description	AREA	DELAY
Bubble sort	179 slices	64.345ns
Batcher's odd-even sort	126 slices	31.630ns
Bitonic merge sort	157 slices	32.436ns
Pipelined bitonic merge sort	154 slices	7.290ns

The table 1 shows the area and delay comparison between the existing system and the proposed system. From the comparison table , we can see the variation of values in terms of area and delay between the existing system and our proposed system.

IV. CONCLUSION

The design and implementation of delay optimized and efficient sorting network is described in this paper. From the obtained results the values are tabulated and from which we can find the improvement in performance of the proposed sorting unit based on speed and the area which we get from the summary . Since the proposed system uses pipeline concept the delay is reduced to 7.290ns which is a very much less when compared to the existing sorting network systems, the area is 154 slices which is less when compared to the bitonic merge sort which occupies 156 slices . But when compared to bitonic odd-even merge sort, the area of pipelined bitonic merge sort is greater , since Batcher's odd-even merge sort occupies just 126 slices. When compared to bubble sort our proposed system is efficient in terms of area and delay. The proposed system has a reduced delay when compared to all other discussed sorting networks which we obtain from the tabulated values. Our results show that there is reduction of 'delay' and 'area', and improvement of the 'speed' in the proposed sorting network.

REFERENCE

[1] R. C. H. Chang, et al., "Implementation of a high-throughput modified merge sort in MIMO detection systems," IEEE Trans. Circuits Syst. I, vol. 61, no. 9, pp. 2730–2737, Sep. 2014.

[2] J. Chhugani, et al., "Efficient implementation of sorting on multi- core SIMD CPU architecture," Proc. VLDB Endow., vol. 1, no. 2, pp. 1313–1324, Aug. 2008.

[3] A. Farmahini-Farahani, A. Gregerson, M. Schulte, and K. Compton, "Modular high-throughput and low-latency sorting units for FPGAs in the large hadron collider," in Proc. IEEE Int. Symp. Appl. Specific Process., Jun. 2011, pp. 38–45.

[4] D. Koch and J. Torresen, "FPGA Sort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting," in Proc. Int. Symp. Field Programmable Gate Arrays, Feb. 2011, pp. 45–54.

[5] K. E. Batcher, "Sorting networks and their applications," in Proc. AFIPS Proc. Spring Joint Computer Conf., 1968, pp. 307–314.

[6] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," IEEE Trans. Comput., vol. 62, no. 7, pp. 1389–1402, Jul. 2013.

[7] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in Proc. ACMISIGDA Int. Symp. Field-Programmable Gate Arrays, 2015, pp. 240–249.

[8] J. Matai, et al., “Resolve: Generation of high-performance sorting architectures from high-level synthesis,” in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, 2016, pp. 195–204.
[9] M. Zuluaga, P. Milder, and M. Peschel, “Streaming sorting networks,” ACM Trans. Design Autom. Electron. Syst., vol. 21, no. 4,pp. 1–30, Jun. 2016.
[10] D. E. Knuth, “Sorting and Searching,” in The Art of Computer Pro-gramming. Reading, MA, USA: Addison-Wesley, 1998.