Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC 2020)
IEEE Xplore Part Number: CFP20V66-ART; ISBN: 978-1-7281-4108-4

Performance analysis of merge sort algorithms

Joella Lobo
Dept of Electronics and Telecommunication, Farmagudi, Goa
Goa college of engineering
Goa, India
joellalobo08@gmail.com

Sonia Kuwelkar
Dept of Electronics and Telecommunication, Farmagudi, Goa
line 2-name of Goa college of engineering
Goa, India
sonia@gec.ac.in

***Abstract*—Sorting of data is employed in numerous applications and plays a vital role in determining the overall performance, speed and power. There is much sorting technique's like the Bubble Sort, Quick Sort, Selection Sort, Insertion Sort, Merge Sort, Heap Sort etc. Out of these, the Merge sort works the best when sorting huge data sets. In this paper, five of these merge sorting algorithms namely serial merge sort, parallel merge sort, bitonic merge sort, odd-even merge sort and the modified merge sort are analyzed and comparison based on resource utilization, delay and area are made. These algorithms are designed and developed to work on FPGA's. They can be parallelized on the FPGA easily to get better performance. Based on the results the best algorithm for an application or problem instance can be selected.**

Keywords— sorting network, FPGA, merge sort algorithm's

I. INTRODUCTION

Most application's which use computing nowadays requires that the data be in a certain kind of order [1]. This procedure to put the data in order is called sorting and is used in various fields [2]. Sorting is used for organizing and filtering the huge amount of data collected [5]. It also shows great significance in modern scientific computing and commercial data processing. Some of the examples are transaction processing, linguistics, combinatorial optimization, genomics, molecular dynamics, weather prediction, astrophysics, etc. For many year's computer sciences is facing computational problems in sorting of a huge amount of data. Many applications having large databases require faster and better-sorting algorithms to yield optimum results. Therefore, parallelism is needed to be increased so that efficient sorting primitives can be implemented.[3][4].

The performance of the database is based on the type of sorting algorithm used. To select the most efficient algorithm, all the information have to need firstly regarding the user's hardware and software available and the comfort of the use and availability of the database. There are some application's which are very crucial and no tradeoffs can be made with regards to performance. For such cases, depending on the requirements new algorithms need to be designed.

Use of sorting in real-time applications is also gaining great significance, for example, real-time graphics scenarios[6], database management systems[7], image processing[8] and numerical simulation [9].

Reconfigurable gate arrays also referred to as field-programmable gate arrays (FPGA), are extensively utilized in the industry for implementing various digital circuits. FPGA which is an integrated circuit uses gate-level parallelism that multi-core GPPs cannot. It has a large memory that can be approached using a few numbers of pins [10]. Using of FPGA become beneficial in the real-time application [11].

In this paper, a comparative performance evaluation of five different merge sorting algorithms is presented: serial, parallel, bitonic, odd-even and modified merge sort.

Given a large number of parallel sorting algorithms, it is a difficult task to select the best algorithm for a particular problem instance. Since there is no identified theoretical model that can be implemented to accurately predict an algorithm's performance, it becomes difficult to make a choice. In the past few years, several implementation studies have been reported in the literature. However, more studies are needed before the point can be approached where a certain algorithm can be recommended for a particular application with any degree of confidence.

II. FIVE SORT ALGORITHMS

When large amounts of data need to be sorted and cannot fit into the main memory of a computing device (usually RAM) it needs to reside within the slower memory which is external to the device (usually a hard drive), in such a situation external sorting is used. Thus, term external sorting is used for a class of sorting algorithms that can manage large amounts of data. It employs a hybrid sort-merge approach. Chunks of data that are small enough to fit in main memory are first read, then sorted, and lastly written out to a temporary file during the sorting phase. In the merge phase, the sorted sub-files are combined into an individual larger file. The merge sort algorithm is an excellent example of external sorting. To start, the file is being split into small pieces such that the size of each piece is small enough to fit into the main memory. Then each piece is sorted individually in the main memory employing a merge sort sorting algorithm. Lastly, the resulting pieces are merged into larger pieces, until the file is completely sorted. Below five of these merge algorithms are described and their time complexities are listed in Table I.

A. Serial merge sort

Merge sort sorting algorithm is a very prominent and efficient sorting algorithm. It's simpler to understand the Merge sort compared to any other divide and conquer

algorithms. Consequently, it is regarded to be a typical representative of such methods and commonly used to introduce the divide and conquer approach itself. Intuitively, merge sort works on an array of 'n' elements as given below:

1. If n<1, the entire array is divided into two halves and these arrays are called sub-arrays which have a size "n/2".

2. Apply merge sort on the sub-arrays;

3. The sub-arrays from step 2 are merged into one sorted array.

Fig 1. below is an example of how the sorting algorithm works for 8 inputs.

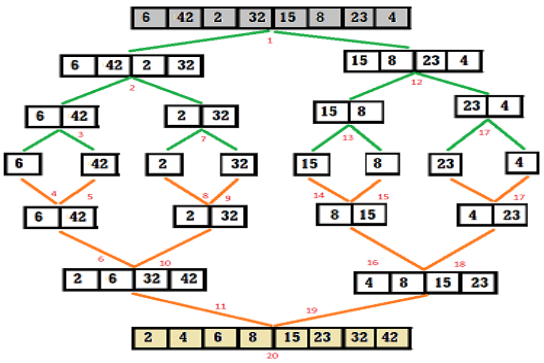


Fig.1: Serial merge, n=8

B. *Parallel merge sort*

With FPGA problems can be solved in a parallel fashion. This sort is very similar to serial merge, the only difference is that here all sub-arrays are executed in parallel. Fig 2 below shows an example of a parallel merge sort. The number of steps required for computation is much lesser than serial merge sort.

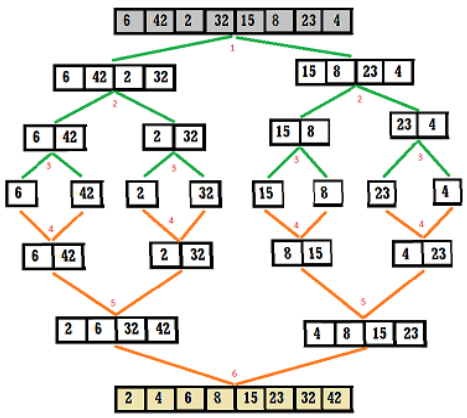


Fig.2: Parallel merge, n=8

C. *Bitonic merge sort*

The bitonic sort is another sorting algorithm. In this algorithm, two halves of its input sequence are sorted in a way such that it creates a bitonic sequence. A bitonic sequence is made up of two subsequence's, one monotonically increases and the second one monotonically decreases. These two subsequences are then further implemented to produce a single monotonic(sorted) sequence.

Batcher's algorithm uses the following steps for sorting an input of 'n' numbers.

1. Convert the n numbers of a given sequence into a bitonic sequence with two subsequence's having n numbers increasing and n numbers in a decreasing.

2. After the bitonic sequence with n numbers is obtained, it is merged into an ordered sequence. Three operations are performed during merging operation which consists of log(n) stages.

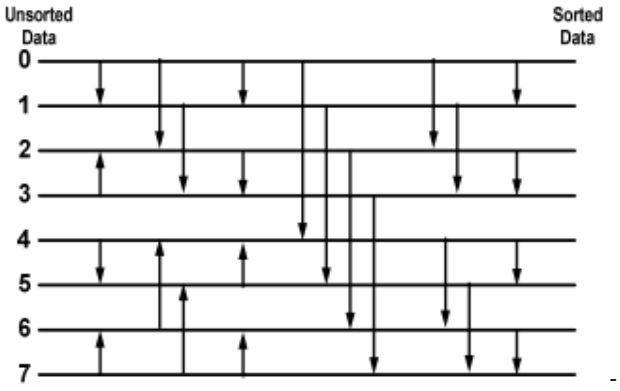


Fig.3. Bitonic merge sort, n=8

D. *Odd-even merge sort*

Batcher proposed an odd-even merge sort algorithm. Here, firstly all the elements with an odd indexed value and even indexed value are individually sorted and later merging takes place. This step is iterated unless a fully sorted sequence is obtained. It is known as an optimal sorting algorithm.

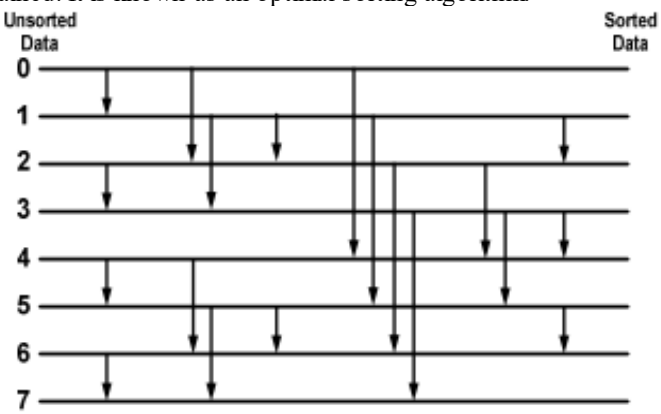


Fig 4. Odd-even merge sort, n=8

E. *Modified merge sort*

A modified merge sorter is made up of a combination of odd-even merge sort and a bitonic merge sort. Its initial circuit consists of an odd-even merge circuit which produces a bitonic sequence and the later part consist of bitonic merge

circuit. The time complexity of the modified merge sort is the same as the odd-even merge sort.

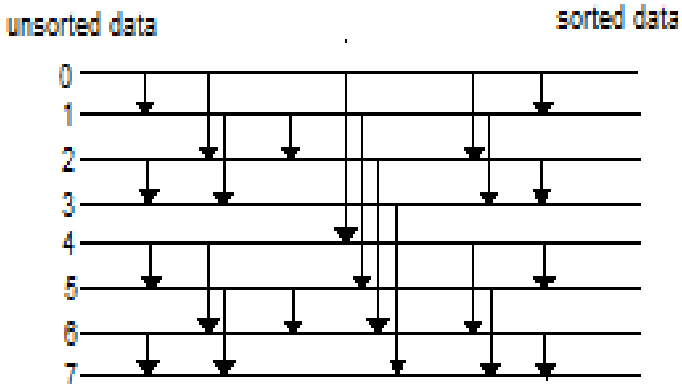


Fig 5. Modified merge sort, n=8

TABLE I. TIME COMPLEXITY COMPARISON

Sort network	Time complexity
Serial merge	O(nlogn)
Parallel merge	O(logn)
Bitonic merge	$O(\log_2 n)^2$
Odd-even merge	$O(\log_2 n)^2$
Modified merge	$O(\log_2 n)^2$

Where n is the number of inputs

III. HARDWARE ARCHITECTURE

A. System Flowchart

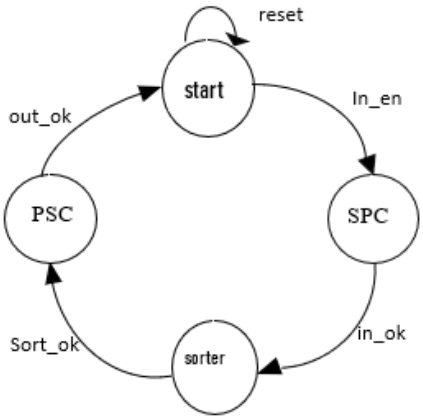
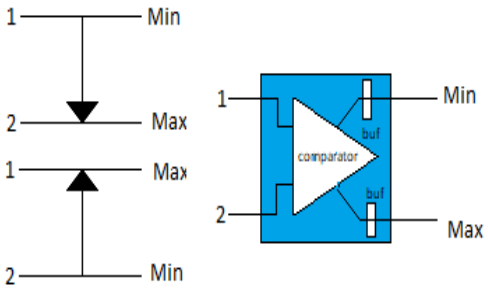


Fig 6. System flow chart of merge sorter

Fig.6 represents how each of the merge sort methodologies is implemented on the FPGA. When the sorter is triggered using the signal "in_en", evaluated as high, instantly the serial-to-parallel converter (SPC), receives the data. It waits until a maximum of 16,32,64 bits of data is collected. The collected data is sent to the sorter only once the signal "in_ok" is forced high. After the data is sorted, if "sort_ok" is resolved as true, these incoming input data are forwarded to the parallel-to-serial converter (PSC).On completion of this step data which is sorted is immoderately produced at the output. Finally, an "out_ok" signal is for the flow to the start and waits for a new set of inputs.

B. Sorter Architecture



(a) (b)

Fig.7(a) Schematic symbol and (b)hardware schematic of comparator

Comparators are used as building blocks in a sorter. A schematic of a simple comparator is shown in fig.7(a), while Fig. 7(b) shows its hardware schematics. A pipelined sorter is implemented in the design which makes use of buffers that stores data at each stage and enhances the speed.

IV. SIMULATION AND IMPLEMENTATION RESULTS

A. Behavioural Simulation

The proposed odd-even merge sorter is written in VHDL. The behavior simulation platform is Xilinx 14.7 and the FPGA used is Vertex-5.

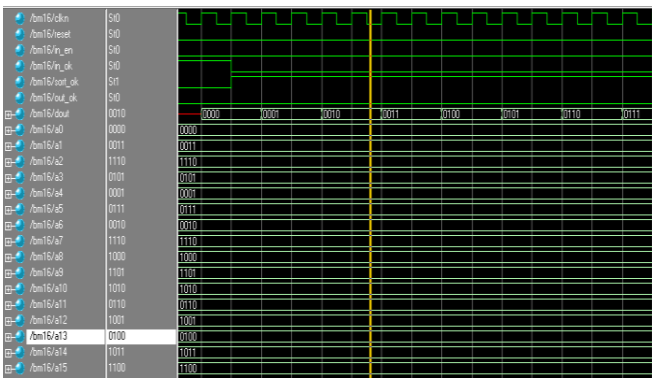
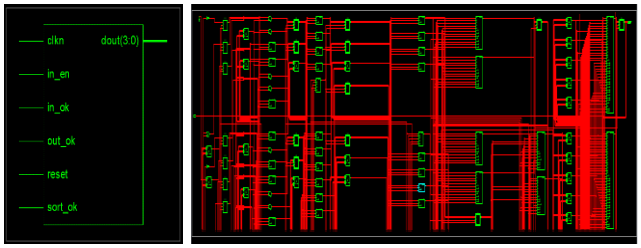


Fig 8. Behavioral simulation of a modified merge sorter

Fig.8 shows a behavioral simulation of a modified merge sorter in which when "in_en=1" a sequence of 16 input data was transmitted. The output sequence is found to be sorted in increasing order. Fig 9(a) and 9(b) show the RTL diagram of a modified merge sorter after the synthesizing process in the Xilinx ise tool.



(a) (b)

Fig.9(a) RTL Schematic (b)hardware schematic of modified merge sorter

B. Performance simulation

To evaluate the merge sort algorithms, resource utilization (number of slices, LUT'S comparators and area constraint ratio) parameters, delay and power are evaluated. The evaluations present the performance metrics based on the different merge algorithms and with varying numbers of input streams to merge.

C. Simulation results

i. Number of comparators used

TABLE II THE NUMBERS OF COMPARATORS REQUIRED FOR EACH ALGORITHM

Sort Type	Input data		
	16	32	64
Serial merge	135	367	927
Parallel merge	135	367	927
Bitonic merge	76	192	448
Odd-Even merge	58	150	300
Modified merge	70	168	348

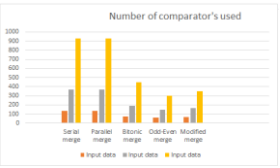


TABLE II. shows the number of comparators used for various sort algorithms along with a graphical representation. The overhead on resource utilization of merging more streams increases because of the larger number of registers and comparators. As, seen an equal number of comparators are used for serial and parallel merge sort but the number decreases for the bitonic, odd-even and modified merge sort.

ii. Number of slices and LUT's

TABLE III. THE NUMBER OF SLICES IN PERCENTAGE REQUIRED FOR EACH ALGORITHM

Sort Type	Input data		
	16	32	64
Serial merge	32%	93%	288%
Parallel merge	31%	92%	286%
Bitonic merge	6%	17%	25%
Odd-Even merge	6%	11%	16%
Modified merge	6%	12%	18%

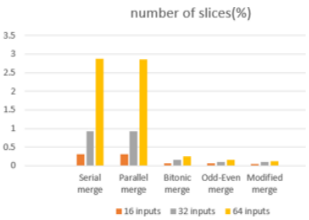


TABLE IV. THE NUMBER OF LUT'S IN PERCENTAGE REQUIRED FOR EACH ALGORITHM(%)

Sort Type	16 inputs	32 inputs	64 inputs
Serial merge	28%	81%	248%
Parallel merge	27%	80%	246%
Bitonic merge	6%	16%	23%
Odd-Even merge	5%	10%	14%
Modified merge	5%	11%	15%

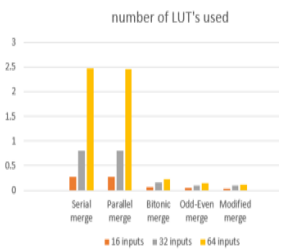


TABLE II and TABLE III show the number of slices and LUT's used. As depicted by the graph the bitonic, odd-even and modified merge algorithms use a small number of slices and LUT's compared to the serial and parallel merge sort.

iii. Area constraint ratio

TABLE V. COMPARISON OF THE AREA CONSTRAINT RATIO

Sort Type	Input data		
	16	32	64
Serial merge	25	87	156
Parallel merge	24	86	154
Bitonic merge	7	24	28
Odd-Even merge	6	13	18
Modified merge	6	14	19

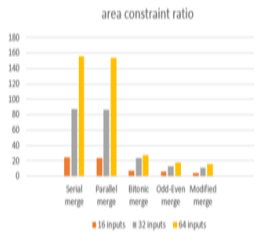


TABLE V. shows the area constraint ratio required for each of the merge sort algorithms. The graph decreases drastically for the bitonic merge sort and further decreases for the odd-even and modified merge sort algorithms.

iv. Timing Delay

A critical path is a path from an input to an output with the largest delay. Xilinx timing analysis tool identifies critical paths. It analyses every path from the input to the output and gives the delay on each path. As shown in TABLE V serial merge sort requires the largest delay while parallel merge delay is reduced to half. The delay for the bitonic merge is slightly more than modified and odd-even merge sort.

TABLE V. SIMULATION RESULTS OF TIMING DELAY IN NANOSECONDS (ns)

Sort Type	Input data		
	16	32	64
Serial merge	23.79	57.79	98.21
Parallel merge	17.57	29.87	59.02
Bitonic merge	2.87	2.931	3.11
Odd-Even merge	2.76	2.72	2.76
Modified merge	2.76	2.73	2.84

v. Power utilization

TABLE VI. SIMULATION RESULTS OF POWER ANALYZER IN WATTS (W)

Sort Type	Input data		
	16	32	64
Serial merge	0.287	0.291	overmapped
Parallel merge	0.282	0.294	overmapped
Bitonic merge	0.268	0.284	0.298
Odd-Even merge	0.265	0.269	0.287
Modified merge	0.266	0.276	0.299

The Xilinx ISE tool estimates the total power consumed (static and dynamic). The results of which are tabulated in TABLE VI

V. CONCLUSION

Selecting the best sorting algorithm for a particular application is a very difficult task. This is based on the tradeoff between resource utilization, speed and area.

It is observed that

- Serial and parallel merge use the highest amount of resource utilization compared to bitonic merge, odd-even merge and modified merge.
- Even though the resource utilization is the same in serial and parallel merge sort, the delay in the parallel merge is much less than serial merge. Implying that parallel execution is much faster than serial.
- The power analysis does not have a significant difference with an increase in input bit size.
- The odd-even and modified merge have a very close value of area used while bitonic merge has a slightly higher value.

Depending on the application appropriate sorting algorithm must be chosen. In future, the merge algorithms can be modified to have lesser resource utilization with faster execution speed. Also, other sort algorithms can be merged with the merge sort algorithm to yield better results.

REFERENCES

[1] Lipu, A. R., Amin, R., Mondal, M. N. I.,& Al Mamun, M. (2016, December). Exploiting parallelism for faster implementation of Bubble sort algorithm using FPGA. In 2016 2nd International Conference on Electrical, Computer & Telecommunication Engineering (ICECTE) (pp. 1-4). IEEE.

[2] D.E Knuth. The Art of computer programming, Sorting and Searching volume II, Addison-Wesley, 2011

[3] Yildiz, Z., Aydin, M., & Yilmaz, G. (2013, November). Parallelization of bitonic sort and radix sort algorithms on many core GPUs. In 2013 International Conference on Electronics, Computer and Computation (ICECCO) (pp. 326-329). IEEE.

[4] Hongyan, C., Junwei, W.,& Xianli, L. (2017, March). Research and implementation of database high performance sorting algorithm with big data. In 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)((pp. 94-99). IEEE.

[5] Kumari, S., & Singh, D. P. (2014, August). A parallel selection sorting algorithm on GPUs using binary search. In 2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014) (pp. 1-6). IEEE.

[6] Bang-Jensen, J., Huang, J., & Ibarra, L. (2007). Recognizing and representing proper interval graphs in parallel using merging and sorting. Discrete Applied Mathematics, 155(4), 442-456.

[7] Sintorn, E., & Assarsson, U. (2008). Fast parallel GPU-sorting using a hybrid algorithm. Journal of Parallel and Distributed Computing, 68(10), 1381-1388.

[8] Inoue, U., Satoh, T., Hayami, H., Takeda, H., Nakamura, T., & Fukuoka, H. (1991). Rinda: a relational database processor with hardware specialized for searching and sorting. IEEE Micro, 11(6), 61-70.

[9] Jeong, S., Lee, Y. M., & Lee, S. (2017). Development of an automatic sorting system for fresh ginsengs by image processing techniques. Human-centric Computing and Information Sciences, 7(1), 41.

[10] Chatter, Mukesh. "High performance self modifying on-the-fly alterable logic FPGA, architecture and method." U.S. Patent 5,838,165, issued November 17, 1998.

[11] Korakoppa, V. P., & Aradhya, H. R. (2017, February). Implementation of highly efficient sorting algorithm for median filtering using FPGA Spartan 6. In 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA) (pp. 253-257). IEEE.

[12] Long, Z.,& Zhang, Z. (2017, October). FPGA-Based Collaborative Hardware Sorting Unit for Embedded Data Processing System. In 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA)(pp. 260-264). IEEE.

[13] Chen, P., Gao, M., Huang, J., Yang, Y., & Zeng, Y. (2018, June). High-Speed Color Sorting Algorithm Based on FPGA Implementation. In 2018 IEEE 27th International Symposium on Industrial Electronics (ISIE) (pp. 235-239). IEEE.

[14] Chang, R. C. H., Wei, M. F., Chen, H. L., Lin, K. H., Chen, H. M., Gao, Y. Y., & Lin, S. C. (2014). Implementation of a high-throughput modified merge sort in MIMO detection systems. IEEE Transactions on Circuits and Systems I: Regular Papers, 61(9), 2730-2737.

[15] Chen, R., & Prasanna, V. K. (2017). Computer generation of high throughput and memory efficient sorting designs on FPGA. IEEE Transactions on Parallel and Distributed Systems, 28(11), 3100-3113.

[16] Nanjesh, B. R., Rajesh, T. H., T. Parallel merge sort based performance evaluation and comparison of MPI and PVM. In 2013 IEEE Conference on Information & Communication Technologies(pp. 530-534). Iejonidhi, M. R., & Kumar, H. A. (2013, April). IEEE.

[17] Colavita, A. A., Cicuttin, A., Fratnik, F., \& Capello, G. (2003). SORTCHIP: A VLSI implementation of a hardware algorithm for continuous data sorting. IEEE Journal of Solid-state circuits, 38(6), 1076-1079.

[18] Yang, Y., Yu, P., & Gan, Y. (2011, July). Experimental study on the five sort algorithms. In 2011 Second International Conference on Mechanic Automation and Control Engineering(pp. 1314-1317). IEEE

[19] Durad, M. H., & Akhtar, M. N. (2014, December). Performance analysis of parallel sorting algorithms using MPI. In *2014 12th International Conference on Frontiers of Information Technology* (pp. 202-207). IEEE.

[20] Harkins, J., El-Ghazawi, T., El-Araby, E., & Huang, M. (2005, December). Performance of sorting algorithms on the SRC 6 reconfigurable computer. In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.* (pp. 295-296). IEEE.

[21] Farmahini-Farahani, Amin, Henry J. Duwe III, Michael J. Schulte, and Katherine Compton. "Modular design of high-throughput, low-latency sorting units." *IEEE Transactions on Computers* 62, no. 7 (2012): 1389-1402

[22] Korakoppa, V. P., & Aradhya, H. R. (2017, February). Implementation of highly efficient sorting algorithm for median filtering using FPGA Spartan 6. In 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA) (pp. 253-257). IEEE.

[23] Tariq, A.S., Amin, R., Mondal, M.N., & Hossain, M.A. (2016). Faster implementation of Booth's algorithm using FPGA. 2016 2nd International Conference on Electrical, Computer \& Telecommunication Engineering (ICECTE), 1-4.

[24] Patel, Y. S., Singh, N. K., & Vashishtha, L. K. (2014, September). Fuse sort algorithm a proposal of divide & conquer based sorting approach with O (nloglogn) time and linear space complexity. In 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC) (pp. 1-6). IEEE.

[25] Alquaied, F. A., Almudaifer, A. I., & AlShaya, M. A. (2011, April). A novel high-speed parallel sorting algorithm based on fpga. In 2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC) (pp. 1-4). IEEE.

[26] Taniar, D., & Rahayu, J. W. (2002). Parallel database sorting. Information Sciences, 146(1-4), 171-219.