INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

FPGA Design for Low Delay Comparison-free, Odd-even Merge Sorter

Preethi[1]*, K G Mohan[2], Jacob Augustine[3], K Sudeendra Kumar[4]

1 Assistant Professor, Department of Computer Science and Engineering, School of Engineering, Presidency University, Rajankunte, Bengaluru, 560064, Karnataka, India
2 Professor, Department of Computer Science and Engineering, GITAM University, NH-207, Nagadenehalli, Doddaballapura, 561203, Karnataka, India
3 Professor, Department of Computer Science and Engineering, School of Engineering, Presidency University, Rajankunte, Bengaluru, 5600642, Karnataka, India
4 Associate Professor, Department of Electronics and Communication, PES University, Banashankari, Bengaluru, 560085, Karnataka, India

OPEN ACCESS

Received: 21-07-2022
Accepted: 04-10-2022
Published: 05-12-2022

Citation: Preethi , Mohan KG, Augustine J, Kumar KS (2022) FPGA Design for Low Delay Comparison-free, Odd-even Merge Sorter. Indian Journal of Science and Technology 15(45): 2458-2467. https://doi.org/10.17485/IJST/v15i45.1384

* **Corresponding author**.

preethisrivathsa@gmail.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Background/Objective: Reduced Instruction Set Computer (RISC) is one of the most common types of architecture involved in microprocessor that has several blocks. There is a lot of scope that can be observed in optimizing these blocks involved in RISC resulting in better and effective microprocessors. **Method:** One of the sub-blocks that plays a prominent role in RISC architecture is sorter, and it can be achieved by modifying the sorting algorithm. **Findings:** A novel odd-even comparison-free sorting that assists in arranging N several data components in roughly N clock cycles is proposed here. N identical blocks are arranged in streamlined manner that are stacked using handful primary logic elements resulting in sorter computation. In the proposed framework, classification and categorization activities are executed in a channeled fashion. The entire design is amalgamated for numerous data sets from imitated indiscriminately generated data elements to all exceptional elements, to all the similar elements, and also from random to completely sorted data elements. It has been observed that, the algorithm appears impartial to the input ordering. **Novelty:** comparison-free unit was implemented on odd-even sorter. Synthesis results indicate that the proposed approach consumes reasonably low FPGA resource. The number of elements consider for sorting was N=8, this architecture takes per element sorting delay as approximately 2.1 to 4.4 ns (1 clock cycle).

Keywords: Comparisonfree technique; Hardware sorting engine; Look ip tables; RISC; Delay

1 Introduction

In recent decades, microprocessors that are commercially available are predominantly employed in the world of embedded systems that are RISC based processors, for instance ARM, Atmel-AVR, and much more are specifically prominent. Processor from the aforementioned families has been used in several applications including IoT devices and energy constrained embedded systems [(1)]. Various sorts of RISC-V instruction sets are developed by numerous research groups across the globe resulting in free accessibility for embedded systems. This is supported by the RISC-V International as successful attempts are made to bring different industries companies throughout the globe, and it is an open for collaboration. The business approach is flexible with open license, with the intention that any person can contribute for the modifications or use the contributions anywhere [(2)]. This has helped the research groups to implement novel methods to modify the RISC-V architecture and result as a hardware accelerator.

RISC-V architecture has various blocks that include CPU, decoder, General Purpose Registers (GPR), Control and Status Register (CSR), ALU, Multiplier, and many other important segments. One of the significant sections of CPU is the prefetch, it is programmed to decipher the commands from memory and exhibits the received instructions to the rest of the processor blocks. In prefetch, sorting is one of the basic blocks to perform the desired operation. Nevertheless, sorting plays a noteworthy role as it assists in processing queries and various kinds of data sorting, the implementation of it can be witnessed in the domain, such as data centres, data mining, IOT edge nodes, and cloud servers [(3)]. Sorting has the primary upper hand in accessing the data effortlessly from the database, as mostly the data required will be in alpha-numerical manner. Hence, an optimized sorting algorithm is essential that can accelerate the performance. Due to this, FPGAs have witnessed its application in intensified domains with complex computations. The basic building blocks of FPGAs are Look-Up Tables (LUTs), memory units, and processing units of small scale to avail the advantage of ease of programming [(4)]. Accelerators are important as it has parallel computing leading to high processing, low latency, low power consumption, and it is observed in diverse applications [(5)].

In recent decades, several sorting algorithms have been developed and tested for diverse applications. The scope of the modified sorter is very wide and it has immense potential to be used for various applications. It has been inspected that verifying the drafted algorithm in a software platform is simpler. However, it is also possible to demonstrate on a hardware platform in which the results will be compared in terms of relative performance assessment, but configuring and executing it is cumbersome. Specifications that are assessed for the comparability with the optimized algorithm are size of data, number of data elements, speed, and time.

Since FPGAs are meant to handle abundant data, on which aggregation, sorting, merging, and joining operations will be performed, resulting in meaningful information. The importance of sorting algorithm is discussed in the earlier section. There are many sorting techniques like Bitonic sort, Odd-even sort, bubble sort, merge sort and insertion sort, which have certain advantages, disadvantages that are discussed below in detail [(6)].

Bubble sort is one among the straightforward and well- known methods [(7)] for sorting elements and provides a viable solution for a small number of inputs. While two numbers are compared, if the numbers are not in order they will be swapped as per requirement, this process will be repeated for many times and in a repetitive manner this step will be implemented for the complete sequence. To complete the sorting, it requires N comparisons in the first iteration for the input of size N, N-1 comparisons in the second iteration, and so on till the elements are sorted. The complexity can be generalized as O(N(N-1)/2) [(8)]. Benefits of this approach are the need of less memory and fewer lines of program to execute; however, the major drawback of this approach is the time required which is higher and it is evident that the algorithm is highly ineffective for large data sets.

Usui.et.al. [(9)] proposed a tree-based merge sorter for large-scale sorting and successfully implemented. It was observed that the performance was degraded for the data that are distorted. To handle massive data sets, a dependable approach was needed. It was achieved by implementing improved sort-merge unit that enhanced the overall throughput but hard to scale for a large Bubble Sort number of data sets. Casper et.al. [(10)] Provides a detailed comparison of various sorting techniques. The parameter that is emphasized sorter that is free from comparator approach. Multiple attempts were made in the recent past to develop comparator-free sorting, illustrating attention towards speed and resource utilization. Similar to this, in the proposed technique along with not using any comparator or complex data path, optimization is also performed [(10)].

Alternative sorter algorithm is Odd-Even method; it is very similar to the bubble sorting with few modifications. The computation is executed until the array elements are sorted and in each iteration two phases occurs, Odd and Even Phases. In the odd phase, we perform a bubble sort on odd, and we perform a bubble sort on even indexed elements [(11)]. This process continues for the alternating index (odd-even pairs, even-odd pairs) until no swapping operation is performed and finally the array is in sorted form. There are two modules involved in this sorting technique: compare and swap. Therefore, the timing complexity can be deduced as $O(N^2)$. By adding parallel computing to this sorting technique, the time complexity of O(N) is possible to be achieved [(12)]. Recently, it is found that this is the pre-eminent sorting method on a hardware device since it presents stable throughput and constant performance over skewed data distributions.

Bitonic sort is based on the Bitonic sequence as well as it is a standard parallel algorithm for sorting. A Bitonic sequence is composed of nearly half of the array in ascending order and another one in descending order[13]. This sorting technique is widely used since the implementation is easy due to its structure compared to odd-even sort. The complexity of this sorting technique is O (N $\log_2$(N)).

Insertion sort is based on finding the right node for inserting a new element by comparing with other elements. Depending on the ascending or descending order, the right node reflects the minimum or maximum element. The data were scanned from right to left in sequential order. The sorting operation is overlapping with the input data activities in this sorting. The major drawback of this sorting method is not suitable for massive data sets.

There are various techniques to sort elements without the comparison unit. An elaborated comparison-free sorting method, technology, and implementation methodology are presented in Table I. From the table, its clearly exhibits the advantages of comparison-free sorting techniques and as well power analysis and reducing delay is not reported in previous work. As already mentioned, odd-even sorting is efficient technique on hardware, a comparison-free unit inside the odd-even sorting reduces delay and provides efficient resource utilization.

The proposed method deals with the design of comparator free sorter and the obtained results are compared with the existing sorters such as Bitonic, Bubble sort and many more. The rest of the paper is organized as follows. Section 2 elaborates the proposed work of comparator-free sorting technique with an example. Section 3 evaluates the performance of the proposed architecture by elaborating the simulation results. Finally, Section 4 concludes the paper.

2 Methodology

The odd-even sorting technique is dedicated for parallel processors. This sorting is based on odd-even and even-odd pairs. The worst-case complexity is O(N^2) and the best-case complexity is O(N). This sorting can be efficiently implemented on FPGA, since FPGAs mitigate the constraint on CPU. This sorting can also achieve higher throughput by implementing on GPU, but FPGAs immerse low power compared to GPUs. The odd-even sorting design follows a hierarchical model i.e. resulting from the instantiation of sub-modules like move through and swap_if_required. Due to modular parameters, it's easy to identify the design of sorting algorithm.

Table 1. Comparison of various comparison-free sorting techniques and methodologies

	Technique	Technology, Implementation Methodology and Complexity	Remarks
[14]	A novel Hardware–based sorting technique which uses SRAM-based memory to store data elements. The stored elements follow the Hamming maximum order representation.	90 nm TSMC technology. A simple matrix multiplication using AND operation sorts the data elements. The Hardware structure mitigates the repetitive flow of data elements between the memory and sorting units. Complexity results in O(N) order.	Power Analysis is not described for the proposed sorting technique. The proposed sorting unit is suitable for single core architecture.
[15]	Sorts input data on-the-fly without any comparators between the data. The proposed technique uses simple registers to hold the binary elements.	A matrix-mapping operation are performed to sort elements. The **one-hot matrix** is transposed to the size N×N and multiplied by the binary matrix of input elements. Thus, eliminating comparison operations on result the sorted elements. Linear Complexity of order O (N).	The proposed algorithm reported a power consumption of 1.6 mW for N = 1024 elements. Not suitable for a large set of elements.
[16]	**Null convention logic** (NCL) unit is proposed. The comparison-free unit use AND-OR gates and one-hot matrix multiplies with the binary matrix.	The NCL sorting unit and comparison free unit compared with respect to transistor, LUTs and power.	By implementing one-hot matrix comparison free sorting, the overall power, area, and time compared to null convention sorting is drastically reduced.
[17]	Proposed a software-based comparison-free sorting method that divides the computations into three phases: count, sum, and sort.	Threads are bundled into single-instruction execution units in each phase, which is a criterion for optimizing hardware use. Based on the mathematical algorithm for CPU implementation (single thread, multi-thread)	Does not concentrate on power

Continued on next page

Table 1 continued

(18)	The proposed architecture, in the first iteration, the largest element (LE) is found. Following that, it identifies the next LE from the remaining data elements in each iteration. The sorting engine has cascaded blocks selected one after another. Each of the blocks in cascaded consists of N number of basic cells Each cell consists of a 2-input AND gate and a tiny switch.	65-nm technology standard cell library. Linear sorting delay of O(N) clock cycles	Test beyond 1024 elements of 32 bits and 2048 elements of 11 bits. Power consumption not discussed
(19)	DAS (Demultiplexer Array Sorting engine) - based on radix-2 sort algorithm combined with counting sort algorithm. It uses bit-serial approaches for bit-unit operations so that their hardware burden is much lighter than that of comparison-based implementations which perform word-unit operations.	DAS uses NXN deMUX array to sort N of k-bit data and completes the sorting operations in k cycles regardless of N. Time complexity of DAS is O(Nk). DAS has $O(N^2)$ hardware complexity due to the array of deMUX,	DAS uses minimal memory because it does not move any data during the sorting operation, but regenerates the input data to memory in sorted order.
(20)	Proposed an FSM that functions as comparison-free sorting algorithm. It's based on counting number of 1's and uses one hot decoder and upcounter for sorting elements.	90 nm CMOS technology In order to reduce hardware complexity, the FSM module is presented with a comparison-free sorting algorithm.	Designed for signed numbers

A novel architecture based on odd-even comparator-free sorting is introduced in this paper. This architecture sorts the N elements in N iterations. The proposed sorting is based on odd-even sorting technique that does not require any comparator or complicated circuitry or intricate algorithm.

The even-odd, indexed data elements are passed to a simple circuitry which consists of basic gates (and, or, not). In the proposed architecture, only basic gates are used, the delay can be ignored and finally results in the improvement of the overall performance.

Algorithm 1 Odd_Even _Comparator_Free
Input: in1, in2, in3, in4, in5
Output: out1, out2, out3, out4, out5
1: All inputs passed to Move_Through
2: a. Move_Through(in1,out1)
b. Swap_If_Required(in2,in3,out2,out3)
c. Swap_If_Required(in4,in5,out4,out5)
3: a. Swap_If_Required(in1,in2,out1,out2)
b. Swap_If_Required(in3,in4,out3,out4)
c. Move_Through(in5,out5)
4: a. Move_Through(in1,out1)
b. Swap_If_Required(in2,in3,out2,out3)
c. Swap_If_Required(in4,in5,out4,out5)
5: a. Swap_If_Required(in1,in2,out1,out2)
b. Swap_If_Required(in3,in4,out3,out4)
c. Move_Through(in5,out5)

Algorithm 2 Move_Through
Input: in
Output: out
out←in

Algorithm 3 Swap_If_Required
Input: ini, inj
Output: outi, outj
1. y=Boolean_Expression(ini, inj)
2. if(y)
a. outj←ini
b. outi←inj
else
a. outj←ini
b. outj←inj

Boolean_Expression: A Boolean expression produces a Boolean value by applying logical functions to the inputs. It plays a major part in building circuits, generating algorithms and as well in programming an application. Hence, minimizing Boolean expression is a prime requisite. There are different methods to minimize the expression like applying algebraic rules, K-map, tabular method, and Quine–McCluskey algorithm (QM). Among these methods, QM algorithm is the most efficient and powerful way of minimizing Boolean expression. In the proposed architecture, the swap decision is made, based on the results obtained by QM method. The prime implicants obtained by the function f are as follows.

If(A>B)
then f=1;
else f=0;
where A, B are inputs to Swap_If_Required function.

Figure 1 shows a schematic block diagram of the proposed architecture. The hardware engine receives the unsorted array as input and outputs a sorted array by applying odd-even comparator free sorting technique. There are major two advantages of this method: N elements are sorted in N-1 clock cycles and delay is very negligible since basic gates are used instead of comparator.

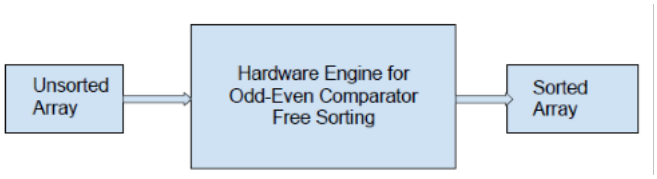

Fig 1. Proposed Architecture

The hardware engine comprises of enumerated blocks connected in a cascaded form as shown in Figure 2. The blocks are selected one after the other. For N sets of n-bit wide arrays, the proposed engine requires N blocks including the first stage carry out move through operation.

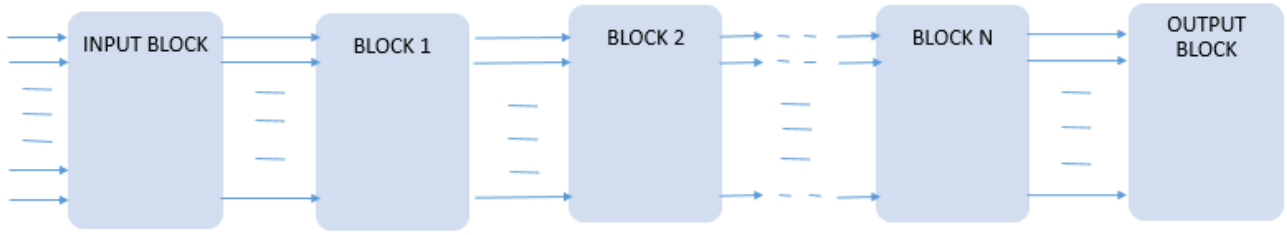

Fig 2. Block diagram of Hardware Engine for Odd-Even Comparator Free Sorting

The internal structure of each alternate block is shown in Figure 3. Each block consists of two main functions: Move through and Swap if required. The Move through function passes the data to the next block without altering the data. The Swap, if required in a block, applies the Boolean expression to the input received. Depending upon the result of Boolean expression which is a comparator free logic, the swapping occurs. The detailed structure of the comparator free block obtained by Boolean expression is shown in Figure 4.

The Boolean expression can be easily solved on FPGAs by using Look Up Table (LUT) and Multiplexor. Hardware LUTs are specially made truth tables that can be customized as per the logic functions required. The values are loaded to FPGA as per requirement. LUTs can be customized for any combinational circuit. In the proposed structure, LUT6 is used. It is a 6-input and 1-output LUT. As shown in Figure 4 there are four LUT6 and three MUX which are used to solve Boolean expressions for comparator free sorting.

Let us consider an example for sorting five data elements, say 4(=0100), 7(=0111), 3(=0011), 5(=0101), and 2(=0010). In the first stage, the first element is moved through the next stage without any changes. The even indexed and odd indexed elements are swapped depending on the value returned by Boolean expression. Here the elements are 7 and 3. The Boolean expression returns 1, hence swapping occurs. The next elements are 5 and 2 need to be swapped. In continuation, this process repeats for the next stage in pipeline. Finally, the last stage has a sorted array.

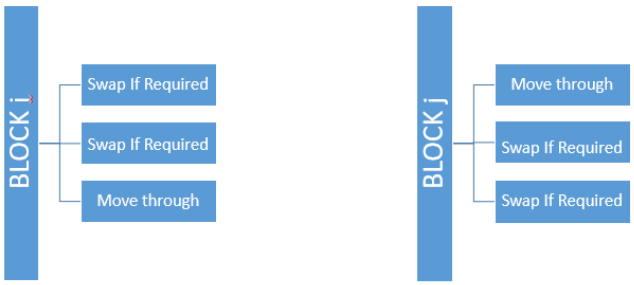

Fig 3. Internal structure of Alternate Blocks. Block$_i$ and Block$_j$

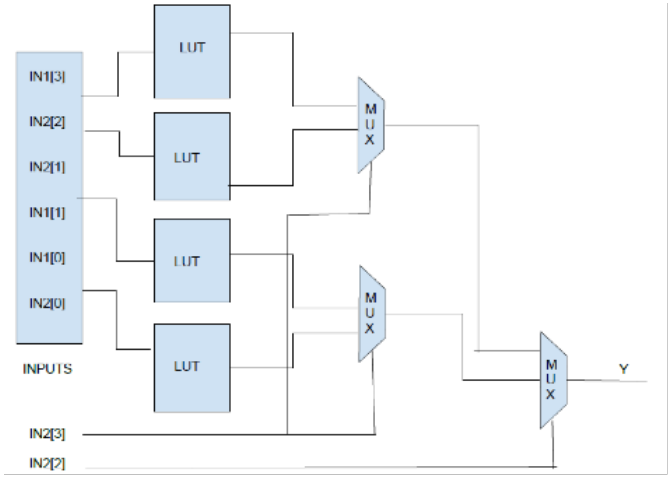

Fig 4. Structure of Comparator Free Block

3 Result and Discussion

The designed sorter algorithmic program was simulated using Xilinx ISE 14.7 on Virtex-7, it is optimized and integrated for 28-nm for the best performance [21], and the complete specification of the board on which it was synthesized is xc7vx415tffg1158-1. The results of the proposed sorter are compared with other sorter methods such as Bitonic, Insertion, Selection, and Bubble with various parameters. The parameters chosen for the comparison are Number of slices, Number of LUT's, Number of logic, Number of LUT Flipflops, IO utilization, and lastly Delay. The results obtained from the chosen comparators and the proposed comparator are tabulated in Table 2. The frequency at which all tests were performed is 224 MHz.

The number of elements considered for sorting was eight. Hence, N=8 in all conditions. Therefore, for 4-bit number the total length is 32-bits, for 8-bit numbers the total length is 64-bits, for 16-bit numbers total length is 128-bits, for 32-bit numbers the total length is 256-bits.

The proposed architecture takes 2.147 ns, 3.104 ns, 3.791 ns, and 4.445 ns delay was observed for sorting 4-bit, 8-bit, 16-bit and 32-bit numbers (respectively) with 8 elements in each case. As well, it is evident that the numbers of LUTs employed by the proposed method/ sorter is also less compared to the other sorting methods. Hence, it can be claimed that along with the reduction of time, the area required will be less. As a result, the authors believe that the proposed method has an upper hand over the compared methods.

Figures 5 and 6 gives the graphical representation of comparison of the various parameters such as Number of Slices, Register, Number of LUTs, Flipflops used, Delay for 4-bit, 8-bit, 16-bit and 32-bit sorters of different approaches.

The Table 3 that follows demonstrate how the comparison-free odd-even unit method reduces LUT count and time delay. By considering the size of input as 8-bits and number of elements=3. The Table 3 represents the comparison between null convention logic unit, existing comparison free sorting unit and Proposed Comparison-free sorter unit. In which comparison free sorting produce the efficient output results. The highlight from table is delay, which is minimal and ignorable.

Table 2. Comparison of various parameters of existing sorters with the proposed sorter for different bit length

Width	Sorter	Number of Slice Registers: (Out of 515200)	Number of Slice LUTs: (Out of 257600)	Number used as Logic (Out of 257600)	Number of LUT Flip Flop pairs used (%)	IO Utilization (350)	Delay: (ns)
4 bits	Bitonic	72	219	221	31	68	2.874
	Proposed Odd Even	**70**	**168**	**168**	**30**	**68**	**2.147**
	Insertion	70	168	168	41	68	2.497
	selection	79	214	216	33	68	3.338
	bubble	76	235	236	31	68	3.433
8 bits	Bitonic	140	430	436	33	134	3.431
	Proposed Odd Even	**134**	**284**	**284**	**28**	**132**	**3.104**
	Insertion	138	637	640	32	132	3.409
	selection	145	378	382	38	132	3.743
	bubble	145	395	400	37	132	3.475
16 Bit	Bitonic	264	650	652	41	260	4.697
	Proposed Odd Even	**262**	**491**	**491**	**37**	**260**	**3.791**
	Insertion	274	1087	1097	45	260	5.607
	selection	270	645	646	42	260	4.865
	bubble	273	678	679	41	260	5.942
32 bits	Bitonic	518	1536	1536	44	516	5.786
	Proposed Odd Even	**521**	**1052**	**1052**	**42**	**518**	**4.445**
	Insertion	620	2777	2867	46	516	5.873
	selection	530	1150	1155	46	56	4.899
	bubble	533	1331	1338	48	516	6.125

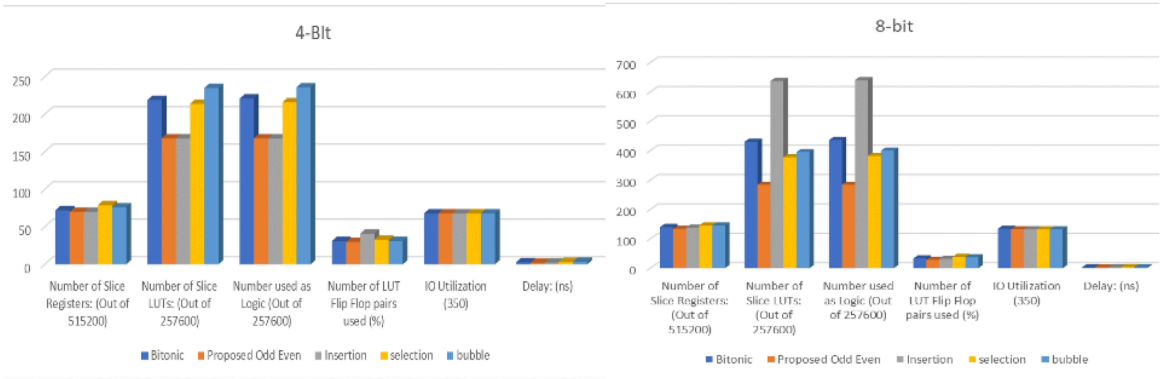

Fig 5. Comparison of the parameters such as Number of Slices,Register, Number of LUTs, Flipflops used, Delay for 4-bit and 8-bit sorters of different approaches

Table 3. Comparison results of NCL, Comparison-free sorter and Proposed Comparison-free sorter unit

Sl. No	Parameter	NCL [16]	Comparison-free sorter unit [16]	Proposed Comparison-free sorter unit
1	Slice	64	49	46
2	LUT	112	85	79
3	Delay	7.646	6.991	1.164

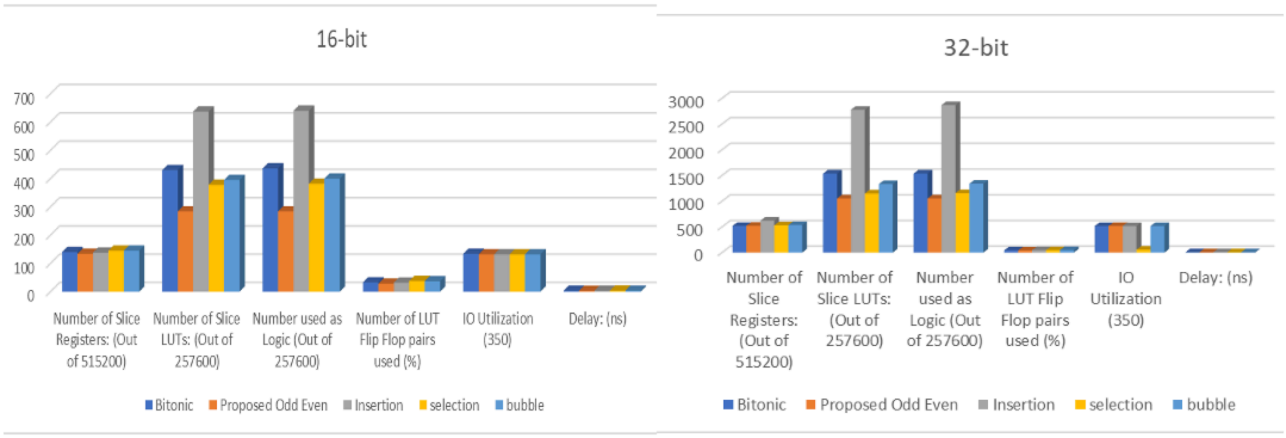

Fig 6. 6. Comparison of the parameters such as Number of Slices, Registers, Number of LUTs, Flipflops used, Delay for 16-bit and 32-bit, different sorters

Table 4 provides comparison of FPGA resource utilization of proposed approach against various recent sorting methods[15,18,22,23]. In this table, proposed architecture consumes less resource for number of elements N=128 and each input size of 32-bits in width. The resource parameter considered for comparison among various sorting methods are Slices utilized, LUTs used and Flip-Flops required in pairs. The device used for simulation varies slightly, however the proposed architecture has high performance in terms of resource utilization. And one more important parameter considered in Table 4 is Delay in nanoseconds for all the sorting methods referred. Comparatively proposed architecture takes less time to sort elements.

Table 5 shows how the proposed sorting engine uses resources when applied to a variety of data sets with different data widths. This table provides detailed resource utilization of proposed architecture. Width of elements considered are 8-bits, 16-bits and 32-bits. Number of elements varied as 256, 512 and 1024 as input sequence. Correspondingly the resource utilized and percentage of resource utilization with respect to slice registers, LUTs and Flip Flops are listed in table.

Table 4. Resource utilization of various sorting methods for 128 elements and 32-bits width

Parameter	[22]	[15]	[23]	[18]	Proposed
Slices Occupied	11142	1665	6832	504	336
LUTs Utilized	71310	3750	43719	2016	1682
Flip-Flops	90169	3330	133584	2016	672
Delay (ns)	400	1500	1600	1300	72
Device	Virtex-7	Virtex-5	Virtex-7	Virtex-5	Virtex-7

Table 5. Resource utilization of comparison-free odd-even sorter architecture

Parameter	Width	Elements (N)	Resource Used	% of Utilization
Slice Registers			4288	0.832
LUTs		256	9088	3.53
Flip Flops			4224	13.51
Slice Registers			8555	1.66
LUTs	8-bits	512	18173	7.05
Flip Flops			8442	27.5
Slice Registers			17145	3.33
LUTs		1024	36347	14.11
Flip Flops			16891	54.03
Slice Registers			8384	1.63
LUTs		256	15712	6.1

Continued on next page

16-bits

Table 5 continued

Flip Flops			8320	26.62
Slice Registers			16762	3.25
LUTs		512	31420	12.2
Flip Flops			13532	43.29
Slice Registers			33531	6.51
LUTs		1024	62844	24.4
Flip Flops			21273	68.05
Slice Registers			16672	3.24
LUTs		256	33664	13.07
Flip Flops			6576	21.04
Slice Registers			33334	6.47
LUTs	32-bits	512	67321	26.13
Flip Flops			20510	65.61
Slice Registers			66682	12.94
LUTs		1024	134655	52.27
Flip Flops			24295	77.72

4 Conclusion

A modified Odd-even approach to sort N elements is proposed. The complexity of the novel sorting is O(N) that is used to denote the sorting speed. The expanded even-odd sorter is tested and verified by implementing on FPGAs that had the configuration of Virtex-7, capable to work at 224 MHz. From the comparison table (Table 2), it is evident that the proposed method has an upper hand in terms of delay, area in terms of LUT and IO utilization. As a consequence of channeling, the sorting and process run in a superimposed manner and hence excludes the need of an increase in memory cycles for classification of the elements. Outcome of the simulation are evident that its minutest resource allocation to counter some of the contemporary hardware-based classification approaches with adequately minor sorting delays. The proposed pipelined structure will be used to harness spatial parallelism in the future to improve the performance for big amounts of data. And one more important parameter considered in Table 4 is Delay in nanoseconds for all the sorting methods referred. Comparatively proposed architecture takes less time to sort elements. Conclusively, the proposed hardware classification approach possibly a useful embedded section as an accelerator for data aware implementation.

References

1) Fang J, Mulder YTB, Hidders JJ, Lee J, Hofstee HP. In-memory database acceleration on FPGAs: a survey. *The VLDB Journal*. 2020;29(1):33–59. Available from: https://link.springer.com/article/10.1007/s00778-019-00581-w.
2) Waterman A, Asanovic K. RISC-V About, RISC-V International. 2020. Available from: https://riscv.org/technical/specifications.
3) Meng X, Yu L, Qin Z. An FPGA-based accelerator platform implements for convolutional neural network. *Proceedings of the 3rd International Conference on High Performance Compilation, Computing and Communications*. 2019. Available from: https://doi.org/10.1145/3318265.3318285.
4) Franklin M, Chamberlain R, Henrichs M, Shands B, White J. An Architecture for Fast Processing of Large Unstructured Data Sets. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors, (ICCD). 2004;p. 280–287. Available from: https://doi.org/10.1109/ICCD.2004.1347934.
5) Preethi DR, G DMK, K DTT. FPGA Based Hardware Accelerator for Data Analytics: An Overview. *SSRN Electronic Journal*. 2019;p. 1–8. Available from: https://dx.doi.org/10.2139/ssrn.3664586.
6) Alif AFMF, Islam SRM, Deb P. Design and implementation of sorting algorithms based on FPGA. In: International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2). 2019;p. 1–4. Available from: https://ieeexplore.ieee.org/document/9036675/similar#similar.
7) Jmaa YB, Atitallah RB. A Comparative Study of Sorting Algorithms with FPGA Acceleration by High Level Synthesis. *Computacion Y Sistemas*. 2019;23(1):213–230. Available from: https://doi.org/10.13053/CyS-23-1-2999.
8) Ali R. Hardware Solution to Sorting Algorithms: A Review. *Turkish Journal of Computer and Mathematics Education*. 2022;13(02):254–272. Available from: https://turcomat.org/index.php/turkbilmat/article/download/12198/8860/21626.
9) Papaphilippou P, Luk W, Brooks C. FLiMS: a Fast Lightweight 2-way Merger for Sorting. *IEEE Transactions on Computers*. 2022. Available from: https://ieeexplore.ieee.org/document/9695338.
10) Ghosh S, Dasgupta S, Ray SS. A Comparison-free Hardware Sorting Engine. *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*. 2019;p. 586–591. Available from: https://ieeexplore.ieee.org/document/8839318.
11) Skliarova I. A Survey of Network-Based Hardware Accelerators. *Electronics*. 2022;11(7):1029. Available from: https://doi.org/10.3390/electronics11071029.
12) Ayoubi R, Istambouli SS, Abbas AW, Akkad G. Hardware Architecture For A Shift-Based Parallel Odd-Even Transposition Sorting Network. In: Fourth International Conference on Advances in Computational Tools for Engineering Applications (ACTEA). IEEE. 2019;p. 1–6. Available from: https://ieeexplore.ieee.org/abstract/document/8851099.
13) Yu L, Jing Z, Yang Y, Tao Y. Fast and Scalable Memristive In-Memory Sorting with Column-Skipping Algorithm. 2022. Available from: https://arxiv.org/pdf/2202.10424.pdf.

14) Abdel-Hafeez S, Gordon-Ross A. A Comparison-Free Sorting Algorithm. In: International SoC Design Conference (ISOCC). 2014. Available from: https://ieeexplore.ieee.org/document/7087612.
15) Abdel-Hafeez S, Gordon-Ross A. An Efficient O(N) Comparison-Free Sorting Algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2017;25(6):1930–1942. Available from: https://doi.org/10.1109/TVLSI.2017.2661746.
16) Gladies EJ. An Efficient Comparison-Free Sorting Unit. *International Journal of Pure and Applied Mathematics*. 2018;118(20):4931–4937. Available from: https://ieeexplore.ieee.org/ielaam/92/7932577/7862290-aam.pdf.
17) Abdel-hafeez S, Gordon-Ross A, Abubaker S. A comparison-free sorting algorithm on CPUs and GPUs. *The Journal of Supercomputing* . 2018;74:6369–6400. Available from: https://doi.org/10.1007/s11227-018-2567-3.
18) Ghosh S, Dasgupta S, Ray SS. A Comparison-free Hardware Sorting Engine. *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2019;p. 586–591. Available from: https://ieeexplore.ieee.org/document/8839318.
19) Yoon M. The VLSI Design of a High-Speed Sorting Engine By Using Demultiplexer Array. *International Journal of Engineering Research and Technology*. 2019;12(6):814–819. Available from: https://www.ripublication.com/irph/ijert19/ijertv12n5_14.pdf.
20) Bhargav TAS, Prabhu EE. Power and Area Efficient FSM with Comparison-Free Sorting Algorithm for Write-Evaluate Phase and Read-Sort Phase. *Communications in Computer and Information Science*. 2019;968:433–442. Available from: https://doi.org/10.1007/978-981-13-5758-9_37.
21) Issartel D, Gao S, Pittet P, Cellier R, Golanski D, Cathelin A, et al. Architecture optimization of SPAD integrated in 28 nm FD-SOI CMOS technology to reduce the DCR. *Solid-State Electronics*. 2022;191:108297. Available from: https://doi.org/10.1016/j.sse.2022.108297.
22) Mashimo S, Van Chu T, Kise K. High-Performance Hardware Merge Sorter. *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2017;p. 1–8. Available from: https://ieeexplore.ieee.org/document/7966636.
23) Rjabov A. Hardware-based systems for partial sorting of streaming data. *2016 15th Biennial Baltic Electronics Conference (BEC)*. 2016;p. 59–62. Available from: https://ieeexplore.ieee.org/document/7743728.