Implementation of a parameterizable sorting network for spatial modulation detection on FPGA

Ismael López Mendoza
Instituto Tecnológico de Sonora
Cd. Obregón, México
ismael.lopezm@outlook.com

José Luis Pizano Escalante
Instituto Tecnológico y de Estudios
Superiores de Occidente
Tlaquepaque, México
luispizano@iteso.mx

Joaquín Cortez González
Instituto Tecnológico de Sonora
Cd. Obregón, México
joaquin.cortez@itson.edu.mx

Omar Humberto Longoria Gándara
Instituto Tecnológico y de Estudios
Superiores de Occidente
Tlaquepaque, México
olongoria@iteso.mx

***Abstract*—In this paper, implementation results of a bitonic-merge sorting network are presented. This is used on a detection algorithm for MIMO Spatial Modulation (SM) communications and it is implemented on an Intel Altera Cyclone IV FPGA. This sorting network has the added peculiarity that it indicates the index or entrance order of the numbers to sort; i.e., each element of the sorted vector has a certain amount of bits concatenated indicative of their original position. Also, as it is completely combinational, the amount of time taken to sort is only the inherent propagation delay, having great performance even if it is not suitable to sort a great quantity of elements. It is possible to implement an entirely parametric sorting network in terms of the number of elements to sort and the bits used for representation of these elements, and the use of resources for index representation is comparable to present implementations.**

***Index Terms*—Bitonic-merge Sorting Network, FPGA, Hardware Architecture.**

I. Introduction

The sorting problem is defined as a sequence of n numbers $\langle a_1, a_2, ..., a_n \rangle$ that permute or reorder as $\langle a'_1, a'_2, ..., a'_n \rangle$ so that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ [1]. There are many applications for classification processes, the most common being: duplicate detection in arrays, priority queues, median and order statistics; other more complex, such as commercial computing (sorting of transactions in banks, of mail by postal code or address, of files by date, etc.), information searching, operations research (application of mathematical models for problem resolution or decision making), event-based simulations, combinatorial search, and numeric computing [2].

Likewise, the semiconductor industry has shown an exponential growth in the last decades regarding hardware complexity and performance. FPGAs are not an exception, even being the ones with the greatest development on this period of time in comparison with the rest of the industry [3].

Given the applications of sorting and the evolution of FPGAs, the interaction between these two has become an interesting research topic.

978-1-7281-3503-8/19/$31.00 ©2019 IEEE

In [4], an arrangement of sorting networks was made (no type specified), working in a sequential way and, according to the authors, is able to sort N data in $N + 2$ clock cycles maximum due to control sequences and the register arrays utilized.

In [5], odd-even merge sorting networks were used. The authors proposed a completely-combinatorial pipelined architecture , and it was proven that this type of architectures are able to dramatically increase the performance of systems and sorting networks in general.

In many applications, it is not necessary to sort all arriving data, but only some of them. That is why [6] proposed an architecture for partial sorting, with the purpose of finding a maximum or minimum subset. This work was based in odd-even merge networks.

In [7], a 16-QAM MIMO 4x4 detector was implemented on FPGA, which also must obtain the minimum value of a data set. In order to achieve this, two sorting schemes were used: merge sort (MS) and winner path expansion (WPE), where, practically, a pipelined sorting network and a smaller sequential sorting network is used, respectively.

These related works confront problems such as sorting speed (throughput), the amount of hardware required for sorting, along with others, even in applications for MIMO detectors in other transmission schemes; but, according to the authors' knowledge, none have studied entrance index carrying in a sorting network.

In this article, results of a sorting network implementation are presented. This is part of the symbol detection process on a MIMO SM (Spatial Modulation) [8], [9] digital communication system; implemented on an FPGA platform. According to the detection problem in [8], it is required to know the minimum value of a data array coming from an earlier processing stage, and here is where the sorting network comes into play. Additionally, the network must indicate the order in which the elements entered, with the purpose of a correct SM detection. The proposed architecture for the sorting network has the characteristic of not being a fixed architecture, i.e., its

dimensions will vary according to the number of transmitting and receiving antennas in the MIMO SM system. In other words, the code for hardware synthesis must be completely parametrizable.

This work is organized as follows: section II presents the structure of the sorting networks; in section III, implementation results and simulation show how the network can be synthesized according to set parameters; and lastly, section IV concludes the paper and discusses future investigation on the subject.

II. Sorting networks

One of the most efficient and traditional ways of sorting in the context of FPGAs are sorting networks. These are attractive for two main reasons: they do not require control instructions, and they are relatively straightforward to parallelize due to the simple data flow [10]. Sorting networks are adequate for short arrays or sequences, whose length is known *a priori*.

The circuits for sorting networks are represented with horizontal wire segments that symbolize each of the elements to sort, and vertical segments that represent comparators between elements. An example of a simple sorting network is shown in figure 1.

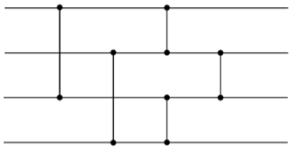


Fig. 1: Example of a simple sorting network.

The unsorted elements enter through the left side of the network, and the ordered output appears on the right side. Comparator modules make a two-element comparison so the smallest element leaves the module through the upper output, and the largest element through the lower output. It should be noted that each wire segment is able to transfer an m-bit number.

There are many methods to systematically build a sorting network. In this article, Kevin Batcher's work and the Bitonic Merging Networks is used, explained in detail in [11]. Figure 2 represents a block architecture to sort eight elements; the arrangement pattern for $Merger[n]$ modules can be seen, where n represents the number of data or elements that the module itself accepts as input.

The internal structure of $Merger[n]$ blocks consists of one $Half{-}Cleaner[n]$ and two $Bitonic{-}Sorter[n/2]$, as shown in figure 3. At the same time, these two are composed, in terms of comparators, as in figure 4.

We can even obtain recursivity expressions for the comparator arrays in figure 4. In $Half - Cleaner[n]$, the inputs i and $n - i + 1$, for $i = 1, 2, ..., n/2$ are compared; in $Bitonic - Sorter[n/2]$, the comparison is made between inputs i and $(n/2^k) + i$, for $i = 1, 2, ..., n/2^k$, where $k = 1, ..., log_2(n)$ and represents each of the groups of comparisons that can be made in parallel, or stages. These stages are illustrated as black boxes in figure 4b, in which two of them are shown, i.e., $k = 1, 2$.

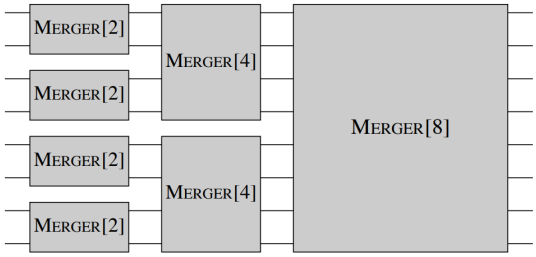


Fig. 2: Network architecture to sort eight elements.

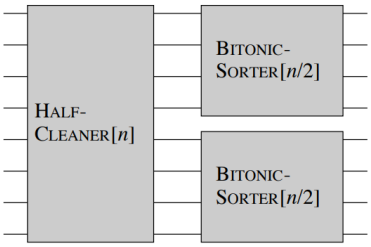
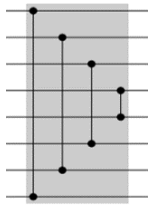
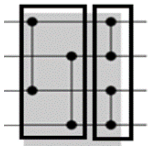


Fig. 3: Internal architecture of $Merger[n]$.



(a) $Half - Cleaner[n]$



(b) $Bitonic - Sorter[n/2]$

Fig. 4: Internal architecture of $Half - Cleaner[n]$ and $Bitonic - Sorter[n/2]$.

In figure 5, a breakdown of figure 2 can be seen with the comparator arrays that form the sorting network, where the $Merger[n]$ and their respective $Half - Cleaner[n]$ and $Bitonic - Sorter[n/2]$ can be identified. Furthermore, a numerical example is presented in order to prove the correct operation of the network according to the zero-one principle, which establishes that if a sorting network is able to sort 1-bit elements, it is capable of sorting elements of any word length as well.

In the example of figure 5, it is seen that an input vector (*10101000*) enters the sorting network, and then goes to the first comparison stage formed by four $Merger[2]$ modules, obtaining the output vector *01010100*. This new vector then

enters the second stage formed by two $Merger[4]$ modules; these at the same time are formed by one $Half-Cleaner[4]$ and two $Bitonic-Sorter[2]$, and the elements leave this stage as *00110001*. The third and last stage is composed by one $Merger[8]$ module, which is formed by one $Half-Cleaner[8]$ and two $Bitonic-Sorter[4]$, and the elements leave the network already sorted as *00000111*. With this example, it is proven that the network is indeed of a sorting type, since the smaller elements left the network through the upper part, while the largest ones left through the lower part.

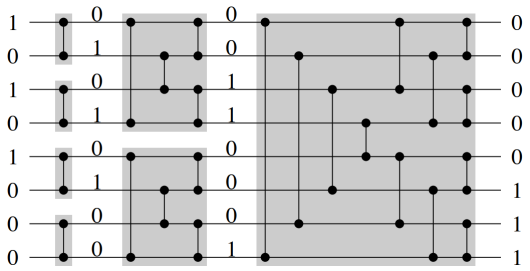


Fig. 5: Network comparator breakdown and example for sorting eight elements.

III. Proposed architecture

Besides from sorting, the network must also indicate in which position in the array the elements were before being sorted; similar to how MATLAB does with its integrated function $[B, I] = sort(A)$ [12], where A is an array with the elements to sort, B is the sorted array, and I is the same size as A and describes the arrangement of the elements in A now in B. For example, if $A = [6, 5, 9, 7]$, then $B = [5, 6, 7, 9]$, and $I = [2, 1, 4, 3]$, so the sorting network synthesized on FPGA must give B and I as outputs.

A. Implementation results

The implementation is made in the software Quartus Prime from Intel Altera, and Modelsim-Altera for verification, everything coded in Verilog HDL. A *top-down* strategy is used for the system architecture. To present the results, the synthesis of $Merger[n]$ modules is shown first, then going down until the basic comparator module.

For parameterization, there are three important parameters to consider, which will determine the dimensions and arrangement of the comparator modules for the correct assemble of the sorting network:

- *ELEMENTS_2_SORT*: indicates the number of elements which are going to be sorted.
- *NUMBITS_NOINDEX*: knowing that the network must also provide the original position index, this parameter indicates the bit word length of each of the elements to sort.
- *NUMBITS*: it is the addition of *NUMBITS_NOINDEX* plus log_2(*ELEMENTS_2_SORT*), i.e., it consists of the total number of bits necessary to give an index to every single element plus the word length of said elements.

The proposed implementation for index delivery is simply concatenating a block of log_2(*ELEMENTS_2_SORT*) bits to each of the elements entering the network so the data transferring is made with *NUMBITS*, even if the comparison is made with *NUMBITS_NOINDEX* bits only, corresponding to the word length.

The highest level of the sorting network is shown in figure 6, and the inputs and outputs of the network are broken down in table I according to the parameters already discussed.

TABLE I:
Breakdown of inputs and outputs of figure 6.

Variable	Type	Size (bits)
Sorting_Elements	Input	*NUMBITS_NOINDEX* × *ELEMENTS_2_SORT*
Elements	Inputs to *Merger* modules	*NUMBITS_NOINDEX*
Index	Inputs to *Merger[2]* modules	$\log_2$ (*ELEMENTS_2_SORT*)
Sorted_Elements	Output	*NUMBITS* × *ELEMENTS_2_SORT*

At the entrance of the network, *Sorting_Elements* (bit vector containing all the elements to sort) is split in *Elements* depending on the amount of elements to sort, i.e., it depends of the *ELEMENTS_2_SORT* parameter. The reasoning for this is due to Verilog language limitations since it does not allow arrays as input, so the mentioned split has to be made manually and directed to the corresponding *Merger[2]* modules.

The *Index* inputs are the bits that are going to be concatenated for the purpose of representing the order in which the elements entered the network. For example, if *ELEMENTS_2_SORT* = 8, the length of these *Index* would be 3 bits that are going to be concatenated to their respective elements.

In figure 7, it is shown how the structure of *Merger* modules in the parameterized synthesis actually matches with the theoretical approach in section II, and table II details inputs and outputs in this module.

TABLE II:
Breakdown of inputs and outputs of figure 7.

Variable	Type	Size (bits)
Elements	Input	*NUMBITS* × *n*
Sorted_Elements	Output	*NUMBITS* × *n*

The most important detail in this table II is that the dimensions of both variables now take into consideration *NUMBITS* instead of *NUMBITS_NOINDEX*, which means that the elements already have their concatenated index bits, and the data transfer through the network will stay this way.

If we continue through the synthesized modules, in figure 8, the necessary comparator arrays for both *Half-Cleaner* and *Bitonic-Sorter* can be seen, and table III shows their inputs and outputs.

For the synthesis of this figure 8, strategies to split the input vector and re-arrangement of elements to their respective

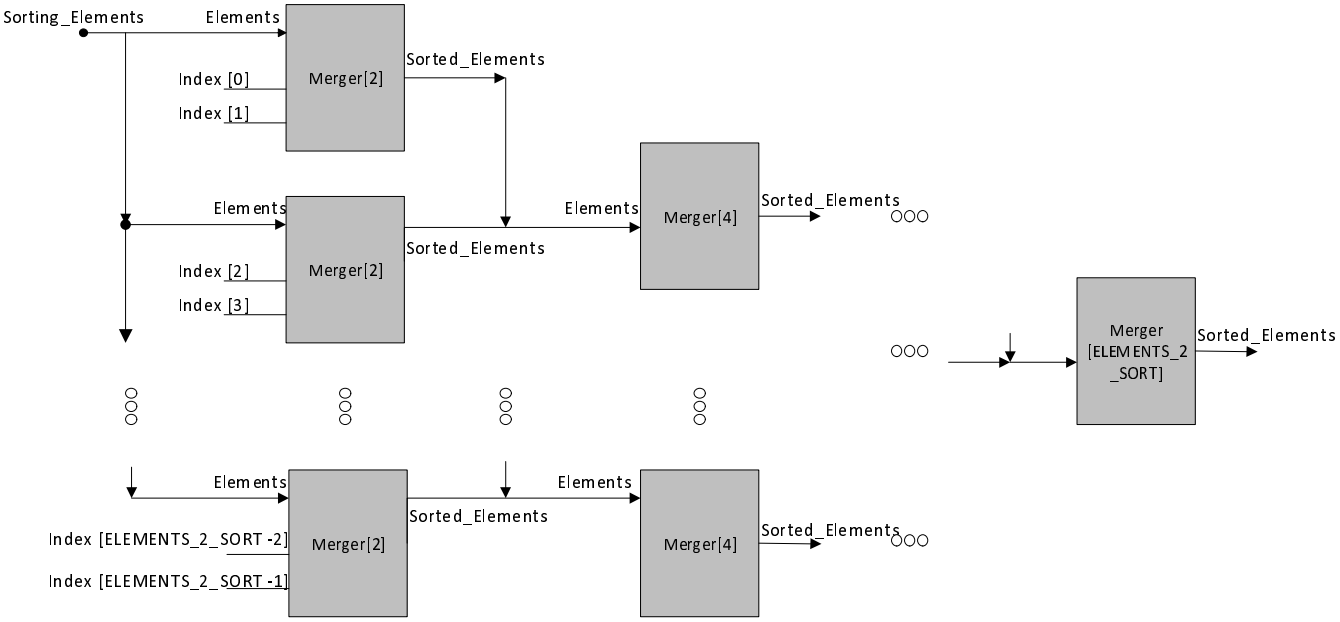


Fig. 6: Structure of the sorting network.

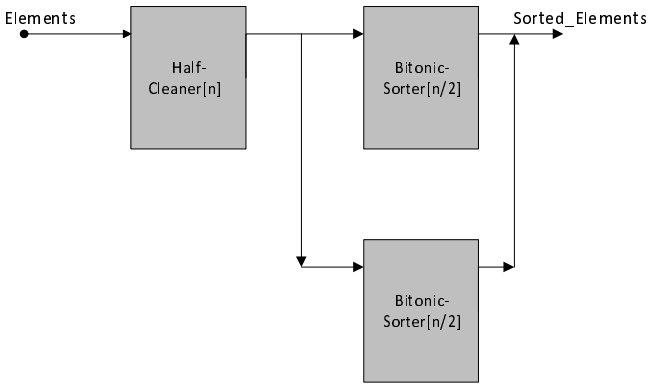


Fig. 7: Structure of *Merger[n]*.

TABLE III:
BREAKDOWN OF INPUTS AND OUTPUTS OF FIGURE 8.

Variable	Type	Size (bits)
Elements	Input	*NUMBITS* × *n*
Cleaned_Elements	Output for *Half-Cleaner*, and input for *Bitonic-Sorter*	*NUMBITS* × *n*
Sorted_Elements	Output	*NUMBITS* × *n*

inputs to modules are used as well, according to what is already established in section II.

Finally, the comparator module, basic unit of the sorting network, is shown in figure 9 and, in table IV, the variables involved on its performance.

In figure 9, one of the peculiarities of this implementation is appreciated: inputs to the module, *A* and *B*, have *NUMBITS* bits of length, but the real comparison is made with

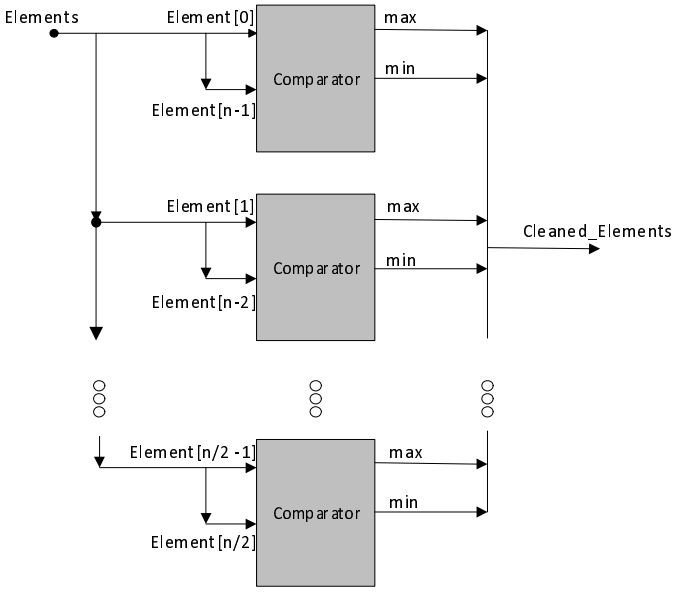
TABLE IV:
BREAKDOWN OF INPUTS AND OUTPUTS OF FIGURE 9.

Variable	Type	Size (bits)
A and B	Inputs	*NUMBITS*
a and b	Inputs to the comparator	*NUMBITS_NOINDEX*
max and min	Outputs	*NUMBITS*

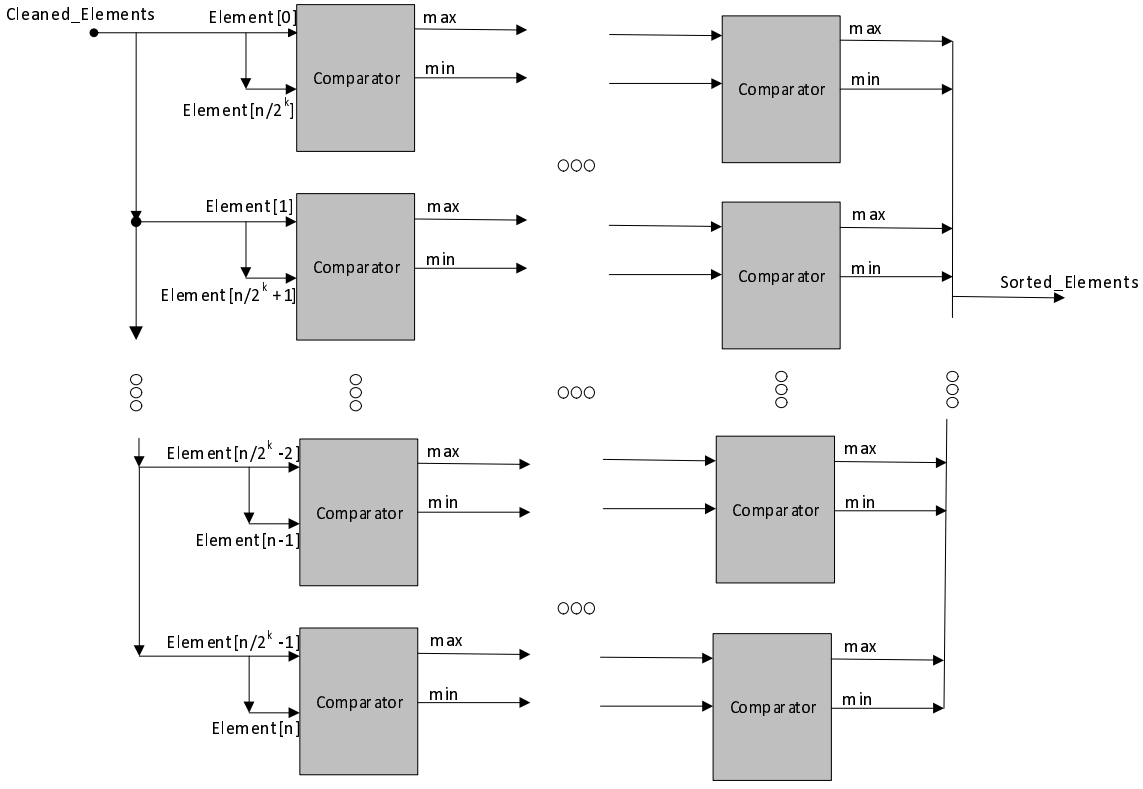
NUMBITS_NOINDEX bits only, corresponding to *a* and *b*. In consequence, when every element leaves the network already sorted, they will have concatenated the index with which they entered, saving this way extra or special processing for index computing, since only a minimum amount of additional hardware is used to handle these added bits.

Now, table V shows a comparative between the resources needed in an FPGA Cyclone IV EP4C5, in terms of logic elements utilized for synthesis of a network capable of sorting a certain amount of data. A comparison is also made between the sorting network with and without indexes, and it is noted that the extra amount of hardware for index transferring along with data does not represent a high percentage of additional elements for synthesis. To emphasize this, the number in parentheses represents the percentage of total logic elements utilized in the chosen Cyclone IV FPGA.

Even if the difference between synthesis with and without indexes is not too considerable, in any case, the resource usage on a sorting network is relatively high in general, and grows exponentially as the number of elements to sort increases. Even so, the performance is very good, since the time it takes the networks to give valid sorted data is the transfer time between gates used in the combinatorial logic; in case the network should grow too much or the clock frequency on the



(a) *Half-Cleaner*



(b) *Bitonic-Sorter*

Fig. 8: Structure of *Half-Cleaner* and *Bitonic-Sorter*.

system where the network is used is high, pipeline could be easily implemented due to the regularity and recursivity of the way the comparator modules are arranged in the sorting network. For example, registers could be placed between every stage, reducing the critical path of the data flow.

B. Simulation results

In order to validate the sorting action of the network, a MATLAB script was used to generate different input vectors which, written in a file, can be read by a Verilog testbench and applied to the sorting network, comparing the results of the MATLAB *sort* function and the hardware synthesis.

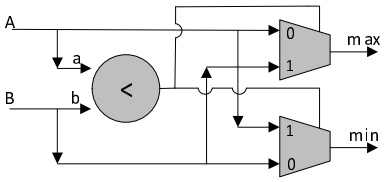


Fig. 9: Structure of the base comparator.

TABLE V:
COMPARATIVE BETWEEN USED RESOURCES FOR DIFFERENT IMPLEMENTED SORTING NETWORKS. 16-BIT WORD LENGTH.

Data to sort (*ELEMENTS_2_SORT*)	**Logic elements (without indexes)**	**Logic elements (with indexes)**
16	3840 (3%)	4257 (4%)
32	11521 (10%)	13121 (11%)
64	32257 (28%)	37697 (33%)
128	86017 (75%)	103041 (90%)

For the sake of space in this article, only one example is shown, where a small sorting network was synthesized, with parameters set as in table VI.

TABLE VI:
PARAMETERS, INPUTS, AND OUTPUTS FOR THE TEST SORTING NETWORK

Variable	**Type**	**Value**
ELEMENTS_2_SORT	Decimal	4
NUMBITS_NOINDEX	Decimal	2
NUMBITS	Decimal	4
Sorting_Elements	Bit vector	01100011

Simulation results are shown in figure 10. Input vector *Sorting_Elements* consists of four elements that, considering MSB ... LSB ordering, are: *11*, *00*, *10*, *01*; entering the network precisely in this order. As mentioned before, each element will have its corresponding index already concatenated while exiting the network; therefore, the observed output vector, *Sorted_Elements*, has as elements: *0100* (value *00* entered with *01* index), *1101* (value *01* entered with *11* index), *1010* (value *10* entered with *10* index), and *0011* (value *11* entered with *00* index), proving the correct functioning of the network as a data sorter.

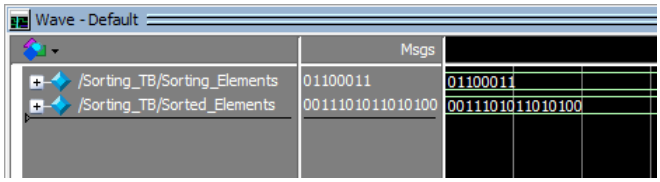


Fig. 10: Results of the test sorting network.

IV. CONCLUSION

Sorting networks are a relatively easy and quick way of generating an architecture to sort data even though they are somewhat expensive in terms of hardware, so this approach is useful if the amount of elements to sort is not too large, just as results show, where the quantity of synthesized hardware explodes with the increase of the elements to sort. Some modifications to the network could be made depending of the specific necessities in each project, just as it was done with the entrance index in this case, denoting versatility.

As future work, a pipelined approach could be considered. Also, techniques for hardware reusing could be a possibility in order to reduce the amount of logic elements used in synthesis, trying not to considerably damage performance on the network.

ACKNOWLEDGMENTS

This work was funded by PROFAPI-2019 and PFCE 2018-ITSON projects.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, third ed., 2009.
[2] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley Professional, fourth ed., 2011.
[3] D. Chen, J. Cong, and P. Pan, "FPGA Design Automation: A Survey," *Foundations and Trends in Electronic Design Automation*, vol. 1, no. 3, pp. 139–169, 2006.
[4] N. Zeraatkar, A. Tavanaie, R. Talebian, and S. Rahimian, "Implementation and synthesis of a sorting network," in *IEEE International Conference on Electronics, Circuits, and Systems*, pp. 1109–1112, 2006.
[5] A. Hematian, S. Chuprat, A. A. Manaf, and N. Parsazadeh, "Zero-Delay FPGA-Based Odd-Even Sorting Network," in *IEEE Symposium on Computers & Informatics (ISCI)*, pp. 128–131, 2013.
[6] A. Rjabov, "Hardware-based systems for partial sorting of streaming data," in *Proceedings of the Biennial Baltic Electronics Conference, BEC*, pp. 59–62, 2016.
[7] X. Wu and J. S. Thompson, "FPGA Design of Fixed-Complexity High-Throughput MIMO Detector based on QRDM Algorithm," in *5th International ICST Conference on Communications and Networking in China*, pp. 1–4, 2010.
[8] R. Mesleh, H. Haas, A. Chang Wook, and Y. Sangboh, "Spatial Modulation - A New Low Complexity Spectral Efficiency Enhancing Technique," in *First International Conference on Communications and Networking in China*, pp. 1–5, 2006.
[9] J. Cortez, C. A. Gutiérrez, F. R. Castillo-Soria, J. M. Luna-Rivera, F. M. Maciel-Barbosa, and E. Ruiz, "Performance Evaluation of Spatial Modulation Techniques in Correlated MIMO V2V Channels," in *2018 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, 2018.
[10] R. Mueller, J. Teubner, and G. Alonso, "Sorting Networks on FPGAs," *The VLDB Journal*, vol. 21, pp. 1–23, 2012.
[11] T. Sauerwald, "I. Sorting Networks," tech. rep., 2015.
[12] MathWorks, "Sort array elements - MATLAB sort." https://www.mathworks.com/help/matlab/ref/sort.html.