Implementation of a General Purpose Sorter on FPGA

Mr.Kamel Alikhan Siddiqui
IEEE Member
Department of Electronics and Instrumentation Engineering
Deccan College of Engineering and Technology, Hyderabad, Inida-500001

Abstract

The objective of the paper is to implement a general purpose sorting algorithm. The paper should offer a sorting network that can be deployed in various applications in impulsive noise reduction filters for image processing and other signal processing applications. The algorithm and sorting network should offer less hardware complexity and better memory usage options. It involves design, simulation and FPGA implementation of a general purpose sorter processor. The paper describes a detailed survey of sorting algorithms that are acquiescent to FPGA implementation, homing in on the most suitable one that may be deployed in digital signal and image processing applications. The work extends by demonstrating the potential of the implemented sorter in noise reduction filters.

Keywords- Digital Algorithm Model & FPGA Implementation, using Xilinx, Finite State Machine for Controller and Design of the Data Path Structure of the Filter

I. INTRODUCTION

The concept of sorting the elements in a sequence of data has become prominent in many application areas. It helps in optimizing the use of various techniques, which requires the data list to be sorted, to perform efficiently. An effective sorting network utilizes an efficient sorting algorithm that put the sequences in a permuted order. The efficiency of different algorithms differs for same applications. Reliable and versatile sorting networks can be implemented using modern computing technology and a solid base of algorithmic knowledge [1]. In this paper such an attempt had been made to design a general purpose sorter processor which can be instigated in a wide set of applications.

Within the scope of the paper, a detailed survey of almost a dozen sorting algorithms has been performed. The suitable one is chosen based on different aspects such as complexity, stability, latency etc. The papers extends by deploying the chosen architecture in various digital and image processing applications such as impulsive noise reduction filters, computational geometry, graph theory and other signal processing areas.

In this context, a quick acquaintance of the working and performance characteristics of sorting networks has been done. The detailed explanation and comparisons of standard algorithms has been given in the following chapters. A sorter basically comprises a network of compare and swap units that will sort all the elements. The overall compare-swap operations mainly depend on the number of elements sorted and not on the values of the elements.

The relative efficiency of the sorting algorithm mainly depends on complexity and run time effectiveness. A sorting algorithm can be framed as an efficient one based on the usage of compare-swap units and it's processing in hardware, for example, parallel or pipelined processing. That means number of compare-swap units and latency are two crucial parameters in any sorting networks.

As in most FPGA designs, the memory usage pattern does matters in all the sorting networks. It is not practical to fit the array correctly into the primary memory. It can exceed the available primary memory and swapping function has to be carried out. Keeping that in mind, it can be decided that it is not the comparison operation that matters, the number of times the memory sections are swapped governs the performance of the algorithm. Thus my search for an efficient sorting algorithm narrows down to one which is data independent and has better latency. The minimum number of compare-swap units that must be executed sequentially defines latency as far as sorting algorithms are concerned.

II. BACKGROUND STUDIES

The background study involved a detailed literature survey of almost all sorting algorithms and their applications. The details of the survey are given below.

A. Sorting Algorithms: Types and application

Sorting is traditionally the process of separating and organizing things according to its type or category. In hardware and software languages it is simply arrange things in ascending or descending order. It can be otherwise defined as sequencing things in particular

order and the output will be a permutation of the input [2]. The various applications of sorting networks include searching algorithms in software, image and digital signal processing etc.

There has been a dozen of sorting algorithms in use for various applications. It is not possible to say which is better than the other, since efficiency of sorting algorithm depends on its applications. This chapter will quickly review some of the prominent sorting algorithms and their practical implications. A detailed explanation of sorting algorithms is given in [1]. There are various factors which categories the sorting algorithms. One major criterion among that is the Big O Notation [2]. It defines the way in which the size of the input data affects the running time and memory. It will estimate the behavioral complexity of any algorithms. Thus to conclude, sorting algorithms are classified mainly on the basis of following factors [2].

1) Efficiency or complexity

This will determine whether the algorithm is in its best, average or worst behavior. The ideal behavior for any algorithm is O (N), where N is the size of the unsorted list. The letter 'O' refers to the big O notation.

2) Stability

Sorting algorithms are said to be stable if they preserve the order of the list with equal key values or indexes.

3) Method of sorting

There are various methods used in sorting algorithms. Comparison sort is one general method. Other than this there are insertion, exchange, selection, merging etc. It can be pointed out that any of these methods can be used alone or in conjunction with each other to make the sorting more efficient.

4) Internal memory bandwidth

Some sorting algorithms posses' in-place memory usage while other needs O (log N) locations to store N elements. These two categories are the less complex ones. The third category requires temporary storage space for data storage and hence possess the worst complexity.

5) Recursive and non-recursive algorithms

Most of the algorithms are either recursive or non-recursive but there are some that may be both.

Above all, in a VLSI design point of view, the simplicity and runtime efficiency in hardware should be the primary constraint in choosing a better sorting algorithm.

A list of commonly used algorithms based on above factors is given below:

Table 1: A list of commonly used algorithms

Nomenclature	*Complexity*		*Stability*	*Memory bandwidth*	*Sorting method*
	Average case	*Worst case*			
Bubble Sort	*$O(N^2)$*		*Stable*	*O(1)*	*Exchanging*
Insertion Sort	*$O(N^2)$*		*Stable*	*O(1)*	*Insertion*
Selection Sort	*$O(N^2)$*		*Not Stable*	*O(1)*	*Selection*
Merge Sort	*O(N log N)*		*Stable*	*O(N)*	*Merging*
Quick Sort	*O(Nlog N)*	*$O(N^2)$*	*Not Stable*	*O(log N)*	*Partitioning*
Binary tree sort	*O(Nlog N)*		*Stable*	*O(N)*	*Insertion*
Heap Sort	*O(Nlog N)*		*Not Stable*	*O(1)*	*Selection*
Shell Sort	*$O(Nlog^2 N)$*		*Not Stable*	*O(1)*	*Insertion*

III. MATLAB IMPLEMENTATION OF MERGE SORT

The main feature of merge sort is that, it will sort a given array of elements by merging two already sorted sub arrays. The function written for merge sort can be recursively use for a large number of elements. The algorithm for merging technique is described below.

A. Merge sort function: Algorithm

- For a given list of unsorted elements, divide the list into two sub lists-A and B.
- Further subdivide the list if necessary and sort the subsequence.
- Merge the two lists by comparing the elements of the sorted list.
- Use the algorithm recursively until full elements are sorted.

In the same way, a matlab code has been written for odd even merge sort to sort a list of large number of elements.

B. Odd-even merge sort: Algorithm

- Divide the list of unsorted elements into halves.
- Sort the two sub lists.
- Take the odd indexed elements of the two sorted list and merge it.
- Similarly take the even indexed elements and sort it.
- Finally merge the odd and even list to get the final sorted array.

To implement the algorithm the already written merge function code can be used. Odd even merge sort is no different from merge sort but it is not data dependent. Thus it can be recursively used as the number of elements increases irrespective of previous results. It reduces the complexity and also reduces the number of compare-swap operations compared to normal merge sort.

The sub lists have been sorted using an inbuilt sort function in matlab. A quick sort implementation is also done in matlab as an experimental attempt. The function for Quick sort has been used in the program in the place of matlab sort function, even though the VLSI implementation does not include quick sort.

C. % matlab code for merge sort –example

```
close all;
clear all;
%N=11;% array containing the list of elements to be sorted.
data=[100 9 2 88 3 99 7 4 31];
N=length(data);
%Checking for data length-odd or even
j=1;
if mod(N,2)==0,
numList=N/2;
else
numList=(N+1)/2;
end
%sublisting the array of elements in data
for i=1:numList
a(j)=data(i);
j=j+1;
end
j=1;
for i=(numList+1):N
b(j)=data(i);
j=j+1;
end
%sorting the sublists
a=sort(a);
b=sort(b);
srt=merge(a,b);
disp(sprintf('Sorted list of given array '));
disp(data);
disp(sprintf('is'));
disp(srt);
```

D. %performing merge sort

```
function [list]=merge(list1,list2)
len_l1=length(list1);
len_l2=length(list2);
%merging the list by comparing the first element of both lists
var=0;
while len_l1>0 && len_l2>0
if list1(1)<=list2(1),
var=var+1;
list(var)=list1(1);
%reassigning the list to define the new first element
for l=2:len_l1
list1(l-1)=list1(l);
end
len_l1=len_l1-1;
else
var=var+1;
list(var)=list2(1);
for m=2:len_l2
list2(m-1)=list2(m);
end
len_l2=len_l2-1;
end
end
%appending the remaining values to the final list
if len_l1 > 0 ,
for p=1:len_l1
var=var+1;
list(var)=list1(p);
end
end
if len_l2 > 0.
for q=1:len_l2
var=var+1;
list(var)=list2(q);
end
end
merge=list;
```

The matlab program for the above example has been provided in the appendix. A matlab code for odd even merging as well as a Quick sort function is also given in the appendix .Following are the results of fixed and floating point simulations.

IV. DESIGN AND ARCHITECTURE OF ALGORITHM

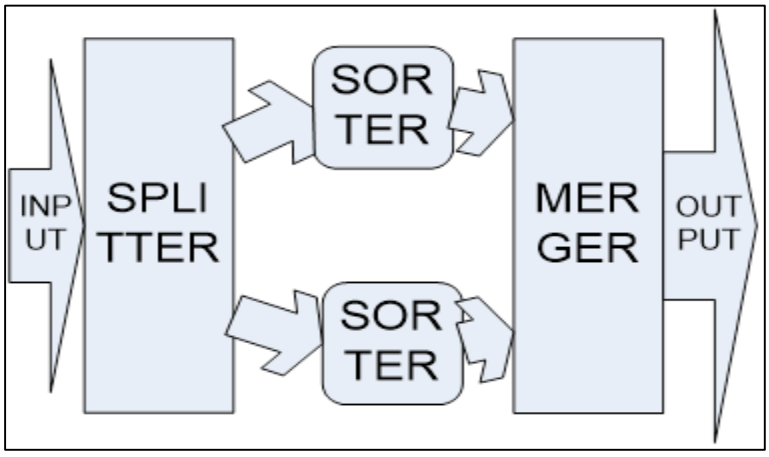

Fig. 1: General schematic of merge function

A. Description of Architecture

The architecture shown in figure 5-2 has been designed for a sorting network that permutes a list of nine elements. The design constitutes a presorted block, followed by a selection logic, which select each input one by one and will give it to the merger unit. The merge unit consist of a comparator unit and a memory to store the inputs. The main advantage of this architecture is that memory usage is independent of intermediate stages and requires in place memory to store the data. With reference to the literatures [4]-[6], this architecture also uses minimum number of clock cycles to sort the data. For nine elements, the sorted output will be provided within ten clock cycles. This is the theoretical assumption, and practicality depends on the environment where the architecture is been implemented.

1) Presorter Unit

The presorter is a combinational circuitry which logically consists of two-input and three-input sorters. Thus to sort four elements two 2-input sorters can be merged together. In order to sort five elements a 2-input sorter and a 3-input sorter has been merged. Thus from the presorter block two sorted list of four and five elements respectively is obtained. The two and three input sorters will compare their inputs and swap it according to ascending order. In effect the basic building blocks of the presorter unit are comparators as in any other sorting networks. The design of the presorter can be done sequentially, but to achieve fast performance a combinational approach has been acquired.

2) Merge Unit

The merge unit performs the core operation in the architecture. It involves the actual merging techniques. Even though merging is involved in the presorter no temporary storage in been implemented and the data will be stored in pipeline. But in merger it exploits the basic idea of merging. It compares the two elements and appends the lowest in the memory location. As this procedure develops the memory location will get filled up. This action happens synchronously so that after nine clock cycles, the full data will be stored in the memory in ascending order. The tenth clock cycle will read out the sorter data.

Storing the values in the memory is achieved by selecting each location using addresses. The address has been provided by an address counter which starts from zero, so that the memory location enabled will be the first one. As the merging proceeds the address location will be incremented until it reaches the ninth location and enabling it. By this time the memory will be full and in the next cycle the sorted list is read out.

Table 2: Signals to and from the sorter chip

Signals	*Type*	*Description*
D0-D8	*Inputs*	*List of unsorted input*
S0-S8	*Outputs*	*List of sorted input*
Clk_L	*Input*	*Single bit System Clock*
Clr_l	*Input*	*Single bit System Reset*

3) Other Blocks and Operations

Apart from presorter and merge unit, the architecture has multiplexers to select input from the presorter to be given to the merge unit. The selection of inputs is such that, for the given sub lists, the lowest of the two arrays will be selected by the two multiplexers and will be given to the comparator unit..

As the comparator compares and assign the lowest in the lowest memory location, the corresponding multiplexer that provided that input, will switch and select the second lowest input. This procedure continues until the multiplexer selects all the inputs. The select line to select each input whenever necessary will be given by a logic circuitry controlled by the system controller. The main advantage of the design is that multiple comparisons are performed simultaneously to make the architecture flexible.

The chapter concludes here with this design and architectural description. In the following chapter attempt has been made to implement the architecture in hardware. The significant component which has not been described in the architectural description is the controller which is the heart of the structure. The success of the architecture mainly lies on the proper design of the controller. The controller design and working has been described in detail in the following chapter.

V. HARDWARE IMPLEMENTATION

This unit explains the FPGA implementation of the chosen algorithm. The architecture is designed to sort an array of nine elements. The input will be parallel given to the sorter chip. It should be noted at this point that implementation in hardware posses some limitations when the memory bandwidth becomes inadequate. Increase the bandwidth is not and never will be a better option. Thus when an array of elements is given parallel the storage options should be considered with great care. The schematics of the design employed in FPGA advantage environment have been described in detail. The top level diagram of the sorter has been described followed by block diagram and explanation of the individual blocks. The simulation results obtained for the blocks have also been

provided which will throw a flash of light on how the architecture works. The compilation and simulation has been performed in ModelSim. Apart from that, a brief explanation has been given on the basic functional blocks such as counters and comparators that are used in the design. An initial latency of 10 cycles is required to get the final output.

A. The Sorter Chip

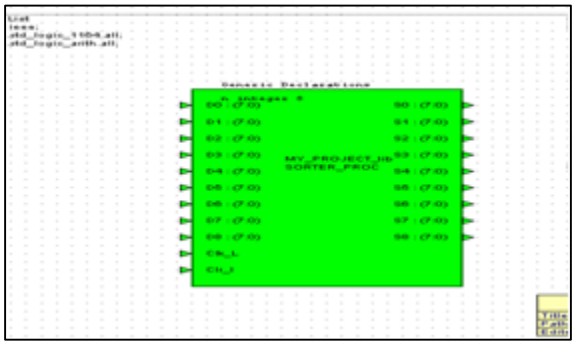

Fig. 2: Sorter Top-level

The inputs and outputs are 8-bits of length. Eight bit vectors are selected based on the application. For median filters intensity values ranges from 0-255, hence should be represented with at least eight bits. A detailed schematic of the block diagram of the sorting network has been shown in figure 3.

The major sub modules in the Figure 6-2 are Presorter, Merger and the Controller. A comprehensive description of each sub modules has been described in the later part of this chapter even though a brief description of all the blocks has been given in this context.

Basically, presorter unit does a number of functions. It splits the array of inputs into sub lists and then sorts it. The function can be also called as premiering. There is a combinational circuitry followed by the presorter, which includes multiplexers. They select the inputs to be given to the merger unit. Merger unit consist of a magnitude detector and memory. The lowest value of the comparator will de stored in the memory sequentially. The controller is the heart of the network. All the control signals including those to the memory are given by the controller. It controls the signal line to the multiplexer so that each signal will be selected according to the output of the comparator in the merge unit. The enable signal to write the data into the memory is also been given by the controller. The controller also sends the clock signal to the address counter which provides the address to the random access memory. The controller designed in the architecture is synchronous with the system clock.

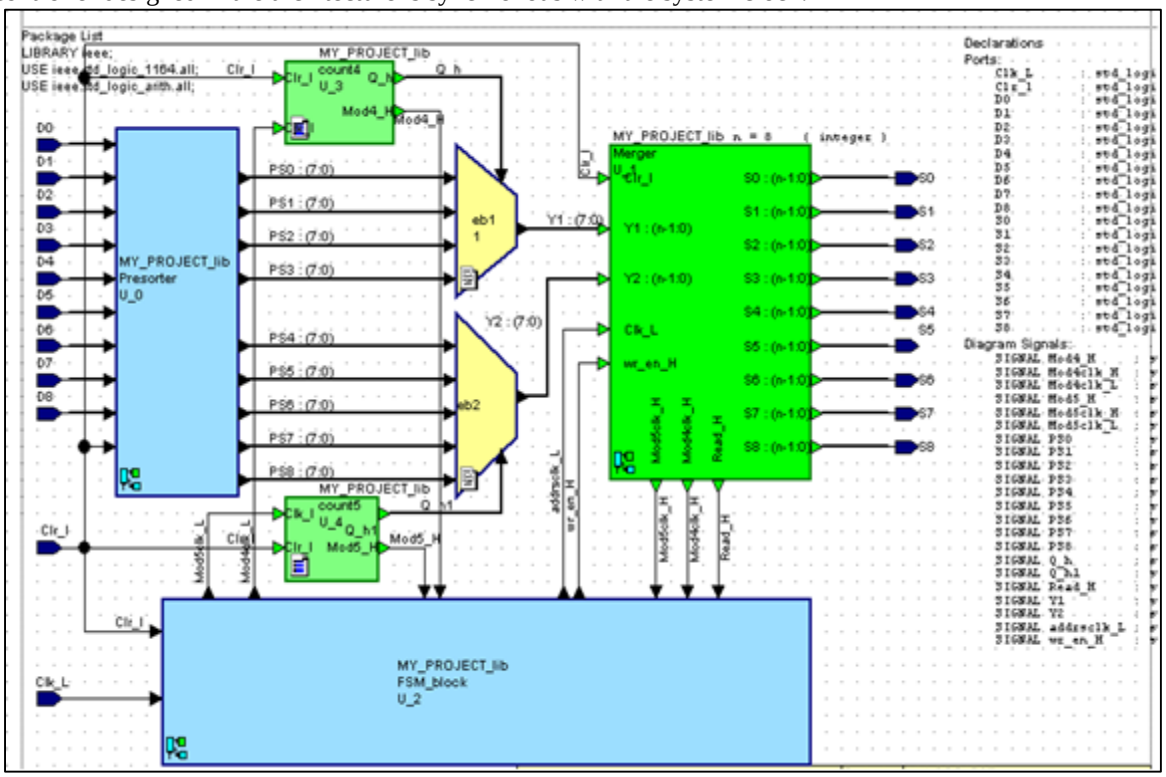

Fig. 3: Block Diagram

VI. CONCLUSION

The main aim of the thesis was to study and analyze the system level implementation of a sorting algorithm and its application in various fields. A detailed survey of almost a dozen algorithms has been performed during the course of the journal which has been apart of the Work. The algorithms have been studied and compared with each other. The algorithm which was found appropriate has been analyzed further from a hardware implementation point of view.

The hardware implementation started with MATLAB modeling. The algorithms under question have been thoroughly verified and tried in matlab. The result obtained paved the way to obtain a clear idea of how it works in FPGA. The next step was to design an architecture which could be implemented using VHDL. The main aim was to come up with an efficient but simple architecture which can be successfully implemented within the time period allotted to the work.

The architectural design was followed by an implementation in the FPGA Advantage platform using VHDL. There have been some hurdles while implementing the design, such as wastage of memory bandwidth, total propagation delay of the data etc, some of which the author was able to overcome successfully. Maximum effort has been put to avoid complexity to the structure and maintain better run time efficiency. The simulation results obtained were analyzed and have been provided in the appropriate chapters. All logic level simulations were done in ModelSim.

As it reached the end of the thesis, testing the architecture has been carried out using a test bench and design for test strategies. The experimentation proceeded to further levels by designing to implement the network in rank order filters. As a beginning MATLAB implementation has been carried out and the results were analyzed. The MATLAB codes and generated HDL codes have been provided in the appendix for reference.

VII. FUTURE SCOPE

Sorting networks and its applications have been a favorite topic among various researchers. It offers a never ending opportunity to experiment lovers. The algorithm which has been proposed in the thesis has got enormous applications in various fields. A great start to future studies on the sorting network has been started in this project itself - Applications in adaptive median filters. The network can be further modified for real time image processing applications and the idea of fixed I/O can be replaced with real time inputs pipelined to the chip. It will reduce the memory bandwidth and pipelining the I/O will result in a higher throughput rate, bearing in mind the initial latency. Opting to provide the data all in parallel without pipelining will result in an I/O bottleneck penalty. This should be considered while designing in that approach.

REFERENCES

[1] Cormen, Thomas,H ,"Introduction to algorithms", Second edition. Cambridge Mass: MIT press c2001.
[2] Knuth, Donald Ervin ,"The art of computer programming", Volume 3 , Sorting and searching, second edition, Reading, Mass. ; Harlow : Addison-Wesley, c1998.
[3] Gonzalez, Rafael .C, Woods, Richard E ,"Digital Image processing", second edition, Upper Saddle River, N.J. ; London : Prentice Hall, c2002.
[4] Mittermair, D, Puschner, P," Which sorting algorithms to choose for hard real-time applications",Ninth EUROMICRO Workshop on Real-Time Systems,1997, 11-13 June, Page(s):250 - 257
[5] Olariu, S., Pinotti, M.C.,Zheng, S.Q.,"How to sort N items using a sorting network of fixed I/O size", IEEE transactions on Parallel and Distributed systems,Volume 10,Issue 5, May 1999 Page(s):487 - 499
[6] Herruzo, Ezequiel.,Ruiz, Guillermo., Benavides, J. Ignacio.,Plata, Oscar.," A New Parallel Sorting Algorithm based on Odd-Even Mergesort", 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing, 2007,Page(s):18-22
[7] Chakrabarti, C.,"Sorting network based architectures for median filters", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Volume 40, Issue 11, Nov. 1993 Page(s):723 - 727
[8] Govindaswamy, V.V.,Balasekaran, G.,Marquis, J., Shirazi, B.A.,"A faster implementation of sequential sorting algorithms using the PARSA methodology", Canadian Conference on Electrical and Computer Engineering, 2003. IEEE CCECE 2003, Volume 2, 4-7 May 2003 Page(s):1313 - 1316
[9] Maheshwari, R.,Rao, S.S.S.P., Poonacha, P.G.,"FPGA implementation of median filter", Tenth International Conference on VLSI Design, 1997. Proceedings. 4-7 Jan. 1997 Page(s):523 – 524
[10] Vasicek, Z., Sekanina, L.,"Novel Hardware Implementation of Adaptive Median Filters", 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008, 16-18 April 2008 Page(s):1 - 6
[11] Fahmy, S.A., Cheung, P.Y.K., Luk, W.,"Novel FPGA-based implementation of median and weighted median filters for image processing", International Conference on Field Programmable Logic and Applications, 2005, 24-26 Aug. 2005.
[12] Morling,R.C.S., Kale,Izzet. ,"VLSI design Techniques",University of Westminster, October 2007.
[13] Douglas L.Perry "VHDL programming by example", fourth edition 2002.
[14] Ashenden, P.J., The Designer's Guide to VHDL, 2nd Edition, Morgan kaufmann, 2001.
[15] Smith, D.J., HDL Chip Design, Doone Publications, 1996