E-ISSN 2581 –7957
P-ISSN 2277 – 3916

CVR Journal of Science and Technology, Volume 16, June 2019
DOI: 10.32377/cvrjst1608

ASIC Implementation of Various Sorting Techniques for Image Processing Applications

Malleswari Akurati
Asst. Professor, CVR College of Engineering/ ECE Department, Hyderabad, India
Email: malli.akurati@gmail.com

Abstract: **The direct implementation of parallel algorithms in hardware is possible with the help of current VLSI technology. The process of arranging the items systematically is known as Sorting. Different meanings of sorting are: ordering: items arrangement in a sequence ordered by using some criterion; categorizing: similar property items grouping. The latest VLSI model analyses the complexity of time. The novel model makes a distinction between "processing" circuits and "memory" circuits; the latter are less important since they are denser and consume less power. This paper addresses the design and analysis of various sorting algorithms, and its VLSI implementation based on a sorting network. The various sorting algorithms are Sinking sort, Merge sort and Library sort; all the three sorting algorithms are compared in terms of area, power and timing with a complete comparison table. Mainly these types of sorting algorithms are used in a real time system; signal processing, image and video processing applications. All the blocks were designed using Verilog HDL, simulated using ncvlog simulator, synthesized in cadence-RTL Compiler and finally implemented in ASIC Encounter using GPDK 45nm technology libraries.**

Index Terms: **Sinking sort, Merge sort, Library sort, RTL compiler and ASIC encounter.**

I. Introduction

In order to obtain high throughput [1] rate, current computers perform several operations simultaneously. Here both the I/O operations and the multiprocessors several computing operations are done concurrently. Such design has to connect various parts of system together (ALU, memory and processor) with a high speed data transferring units. Generally cross-bar switching is used for this, but for large number of inputs (m X n matrix) requires large hardware and power. This paper describes the fast ordering networks. As the new generation computing systems are having high performance, the basic elements like Algorithm-structured chips are helpful for better performance.

- While fabricating the VLSI circuit, the cost effective factors like silicon area places an important rule.
- The circuit area always depends on the logic size and also the architecture modularity;
- The main parameter which effects the network is the circuit speed;
- When talking about different types of sorting algorithms [3], the other parameters like area of the chip and sorting time are also need to be considered.
- Here various sorting techniques are discussed and compare them in various aspects like time, area, power and complexity;

The major contribution of this paper is to describe the basic approach of VLSI sorting device. The main aim is to reduce the complexity in all aspects.

II. Sorting Techniques

A. Introduction

In ASICs, there are more traditional approaches to perform sorting to achieve high throughput and low latency. Sorting networks became more popular and impressive due to the following reasons. First one is pairs are not required for branch type instructions i.e. loop instructions. The other one is due to the concept of instruction level parallelism. Mainly, these types of networks are performed when their data size or bus width size is less. Mostly in Intel or Pentium processors, these types of networks generally use Vector primitives which have been studied from distributive computing[10].The circuit is mainly designed using horizontal intersection or inter connective wires and vertical elements. i.e. comparators. Knuth notation is mainly used and focused to design each element in the comparator[3].The unsorted elements are applied at the left, one element per wire as shown in Figure 1. It describes various sorting techniques separately by its pictorial representation. The extreme right side of the design represents sorted output. Compare and swap stages of the network are connected through intermediate wires. m-bit number of sorted elements of data are transferred through the intermediate wires. The two element sort is performed to the compared elements in which the smaller values left on the right side and the larger values left on the lower side.

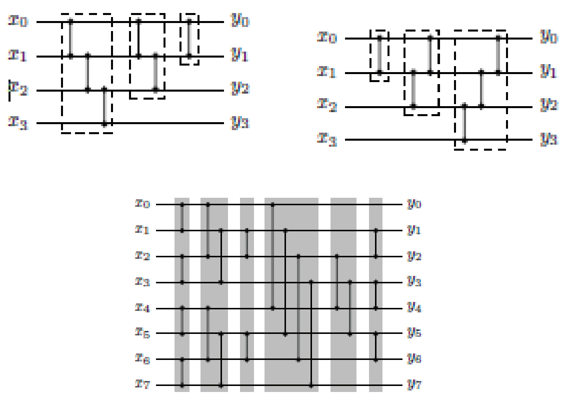

Figure 1. Sinking Sort, Library Sort and Merge Sort

B. *Time Complexity*

By comparing various sorting algorithms, the choosing of prescribed sorting algorithms or network for a dedicated application makes the network simpler. The throughput of a sorting network is analyzed mainly due to the timing factor of the algorithm. The Robustness of a network reflects the relative throughput and it is usually given in the form of notation Big-D. Here, D represents the network Robustness and p represents the size of the pair of the network. Table I represents and gives a basic idea about the robustness of multiple sorting networks.

TABLE I
COMPLEXITIES OF VARIOUS SORTING ALGORITHMS

Types of sorts	Time Complexity		
	Average	Best	Worst
Sinking Sort	D(p^2)	D(p^2)	D(p^2)
Selection Sort	D(p^2)	D(p^2)	D(p^2)
Library Sort	D(p^2)	D(p)	D(p^2)
Merge Sort	D(p log(p))	D(p log(p))	D(p log(p))

C. *Sinking Sort*

It is a simple sorting algorithm. The other name of sinking sort is **Bubble sort.** This is also referred to as Comparison sort as it compares smaller and larger elements in the list. In this type of sorting, it continuously goes through the sorting list, where each pair of the adjacent list is compared. If they are in the correct order, it would not change the list and if the order is wrong, the items will be swapped. The same process is repeated until the items are in the same order. i.e. until further swapping is not required. The advantage of this type of sorting technique is very simple and the drawback is slow and practically not useful for most of the problems compared to insertion sort. Practical implementation of bubble sort is possible when the input is generally in sorted order but may usually have some out-of-order elements nearly in position. [5]

Algorithm

```
Procedure sinking sort (S: sortable elements list)
p=length (S)

repeat
        swapped = false
        for k=1 to p-1 inclusive do
              if  S[k-1]> S[k] then
                      swap (S[k-1], S[k] )
                      swapped =true
                      end if
              end for
      until not swapped
end procedure
```

Among all the sorting algorithms, sinking sorting is the simplest one in understanding and implementation point of view. But even compared to the Library sort, it is inefficient as the efficiency decreases adequately due the complexity of D (p^2)

D. *Library Sort*

The other name of Library sort is Insertion sort. Always iteration is done on one input element, and growing a sorted output list. In this type of sorting, one element is removed from the input data at each iteration and after finding the location of the current element it places the element in that respective position within the given sorted list[6]. This process is repeated until there are no such elements to sort. The current value is always compared with the largest value in the sorted list. The position always depends on whether the element is smaller or larger. If the elements value is small, it searches for its correct position and places it in the particular place within the sorted list. Otherwise it ignores that element and moves to the next element [6].

Algorithm

Based on the insertion sort algorithm only, the proposed sorting is done. In this algorithm, always it searches for the correct position of elements and it inserts all the input elements in the correct order. The pseudo-code representation of this algorithm is as follows:

Algorithm

```
 Function Insert Sort
        for each unsorted C{
        k=0;
                while (k<p) and (C>M[k]) ) {
                        M[k]=M[k+1];
                        k=k+1;
                }
        [k-1]=C;}
end function;
```

The vector M whose length is infinite is considered and the input data is entered into this vector. But generally it is not possible. So that the option deletion have to be used [8]. Every time, the smallest value in that list is eliminated, meanwhile [9], the data which is going to be deleted should be indicated by the outer input signal.

E. *Merge Sort*

A Merge sort follows divide and conquer algorithm conceptually and the working procedure of it is as follows:

1. The unsorted list is divided into p sub lists and each sub list containing one element (only one element list is taken and sorted).
2. Every time it combines sub lists so that new sorted list is generated until there is only one sub list left in the given list [7].

Algorithm

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge function is used for merging two halves. The merge (arr, 1, a, q) is key process that assumes that arr [1..a] and arr [N+1..q] are sorted and merges the two sorted sub-arrays into one. [7, 8]

```
Merge sort( arr [],1,q)
If q>1
Find the middle point to divide the array into two halves:
        Middle a= (1+q)/2.
Call merge sort for first half:
        Call merge sort ( arr, 1 ,a).
Call merge sort for second half
        Call merge sort (arr,a+1,q)
Merge two halves sorted in step 2 and step 3
        Call merge (arr,1,a,q)
```

Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

T (p) = 2T (p/2) + D (p).

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is D (p log p). [9]

Time complexity of Merge Sort is D(p log p) in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

III. Implementation And Result Analysis

A. Sinking Sort

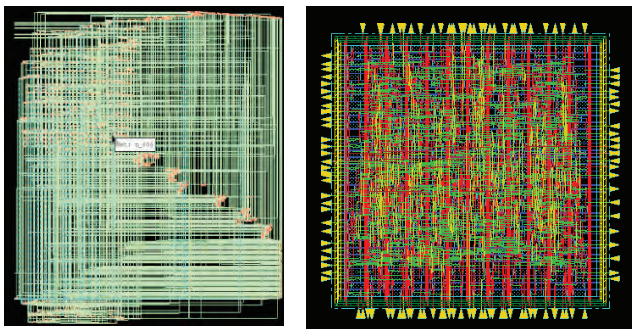

Figure 2 (a), (b). RTL Schematic and ASIC Implementation of Sinking Sort

B. Library Sort

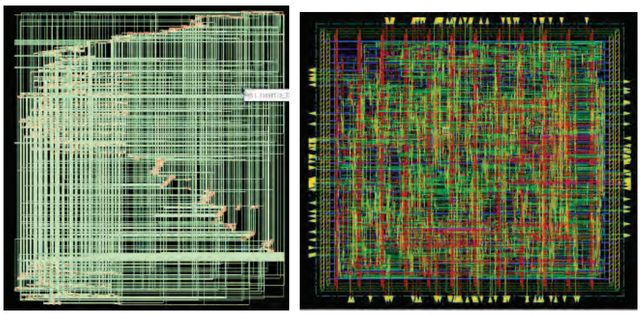

Figure 3 (a), (b). RTL Schematic and ASIC Implementation of Library Sort

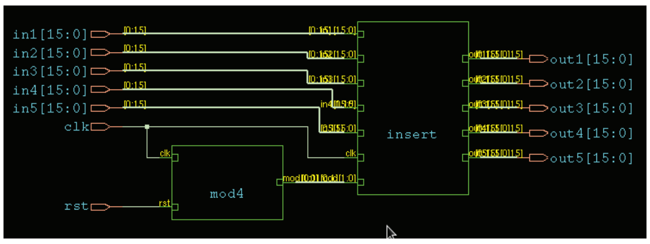

Figure 4. Top Level Design of Library Sort

C. Merge Sort

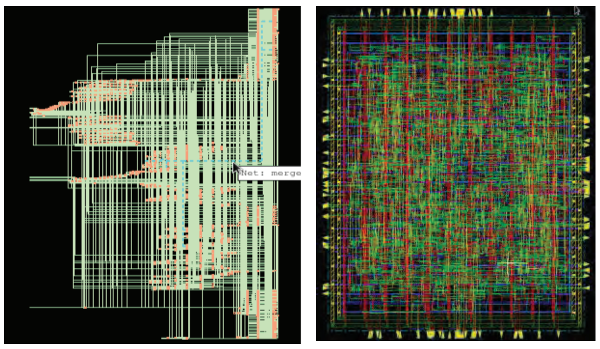

Figure 5 (a), (b). RTL Schematic and ASIC Implementation of Merge Sort

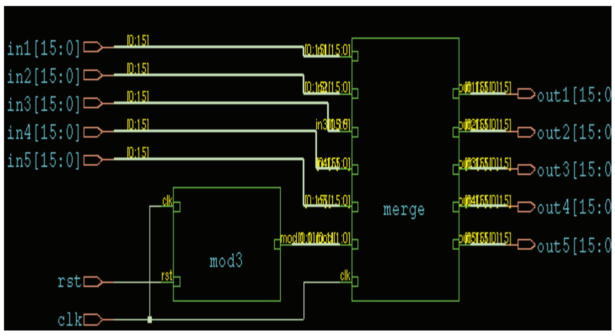

Figure 6. Top Level Design of Merge Sort

Section III mainly deals with the Implementation and Result Analysis of various sorting techniques. The HDL language used and tools used are mentioned below.

HDL Language Used: Verilog HDL
Simulation Tool: Ncvlog Simulator
Synthesis Tool: RTL Compiler
Physical Design Tool: ASIC Encounter

Figure 2(a), (b) represents the RTL Schematic and ASIC Implementation of Bubble Sort. It is very flexible to design on an IC due to its sorting network. Figure 3(a),(b) describes the implementation of Library sort based on its sorting technique. Internally Library sort consists Mod-4 counter to count the number of moves or paths of the respective node in the network which is shown in the figure 4. Figure 5 (a), (b) gives implementation results of Merge Sort. Merge sort internally consists of Mod-3 counter to count the number of elements replaced or eleminated from the sorting newtork respectively as shown in the figure 6. All the sorting networks are taken in the form of m * n matrix only to have equal distribution in the network. Timing of the network is calculated interms of nanoseconds and power is calculated interms of nanowatts and area is explained and derived interms of cells which are occupied on an IC. The Table II gives the comparative analysis of various sorting techniques i.e., Sinking Sort, Library Sort and Merge Sort in the respective aspects such as Area, Timing and Power. Power analysis is furthur calculated interms of Internal Power, Switching Power and leakage power. Compared to all the above mentioned Sorting Techniques [9] Merge Sort will be more efficient in all the aspects and it is most preferred. Arrangement or ordering of elements in the network will be performed very fastly compared to Sinking sort, Selection Sort and Library Sort.

IV. Comparison Between Sorting Techniques

A. *Area, Speed and Complexity*

- To compare different VLSI algorithms and architectures for sorting, it is of great interest to take a look at the lower bounds of area or speed, in the sense that, cannot solve a given VLSI problem using less than a lower bound of silicon area, or less than in a given amount of time. Because of the trade-off between area and speed, it is also important to consider lower bounds of the product ST, or of ST^2.
- Computational Complexity [10](worst, average and best behavior) in terms of the size of the list (p). For typical serial sorting algorithms, good behavior is D(p log p), with parallel sort in D($\log^2$p), and bad behavior is D(p^2). Ideal behavior for a serial sort is D(p), but this is not possible in the average case. Optimal parallel sorting is D(log p). Comparison _based_ sorting algorithm need at least Ω(p log p) comparisons for more inputs.

B. *Comparison Table*

TABLE II.
COMPARISON OF SINKING SORT, LIBRARY SORT AND MERGE SORT IN TERMS OF SETUP TIME, HOLD TIME, AREA AND POWER FROM THE ANALYSIS OF ASIC IMPLEMENTATION.

Parameters		**Bubble Sort**	**Insertion Sort**	**Merge Sort**
Timing Analysis	Slack (ns)	7923	7464	5184
Total Area	Top Module	1254 Cells, 2678.89 area	937 Cells, 1951.79 area	1000 Cells, 2302.69 area
Total Power	Internal Power	50.8644%	49.4750%	57.1171%
	Switching Power	49.1194%	50.4689%	42.8407%
	Leakage Power	0.0463%	0.0561%	0.0422%

V. Conclusions

This paper gives the importance of SoCs /ASICs to accelerate ordering. It presents the various ways to fabricate sorting networks on ASICs and briefly explains the on-die resource utilization. For synchronous fally-pipelined implementation, the flip-flop and LUT utilization of a circuit shows more impact and complexity on the hardware. In the aspect of multi core systems, ASIC shows how the data can be accessed internally at each coprocessor stage. This paper also gives various types of data processing operations where ASICs have multiple advantages i.e. parallelism-pipelining and low latency. Many ways are discussed in which ASICs /SoCs can be embed into a large system so that the performance can be increased rapidly. This type of approach leads to high performance on ASICs in terms of efficiency and latency. Moreover, the main agenda is to achieve high performance of the network. It is challenging to maintain this performance, once the hardware implementation of the algorithm is integrated into a full system. Next to raw performance, these experiments also show that ASIC brings additional advantage in terms of power consumption. Because of these things, ASIC plays an important role in heterogeneous - core architectures. The work reported in this paper is to incorporate the capabilities of ASICs into data processing engines in an efficient manner.

References

[1] C.D.Thompson, ”A complexity theory for VLSI,” Ph.D,dissertation CMU-CS-80-140, computer science department,Carnegie-MellonUniversity, Pittsburgh,PA,August (1980).
[2] D. E. Muller and F .P. Preparata, ”Bounds to complexities of networks for sorting and for switching,” J.Ass. Computer Mach.,vol. 22, pp.195-201, Apr. 1975.
[3] Ajai, M.,Komlos, J. and Szemeredi, E. (1983) , “Sorting in clog n parallel steps,” Combinatorica, 3, 1-19
[4] Bildari, G. and Preparata, F.P. (1985). “A minimum area VLSI network for O(log n) time sorting, ”) IEEE Transactions on computers, C-34,336-343.
[5] Batcher, M. E. (1968), “Sorting networks and their applications,” Proceedings of AFIPS Conference, pp.307-314
[6] D. E. Knuth, “The Art of computer programming,” Vol. 3: Sorting and Searching,” Reading, MA:Addison-Wesley,1973
[7] S. Todd, “Algorithm and hardware for a merge sort using multiple processors,” IBM J. Res.Develop., vol. 22, pp. 509-517, Sept. 1978
[8] J. Vuillemin, “A combinational limit to the computing power of VLSI circuits,” IEEE Trans. Comput., vol. C-32, pp. 294-300, Mar. 1983.
[9] L. E. Winslow and Y. C. Chow, “Parallel sorting machines: Their speed and efficiency,” in Proc.AFIPS 1981 Nat. Comput. Conf., Fall , pp. 163-165, 1981
[10] A. C. Yao, “Some complexity questions related to distributive computing,” in Proc. I lth Annu. ACMSymp. Therory Computt., pp. 209-213, May 1979