A Comparison of Three Representative Hardware Sorting Units

Rui Marcelino[1], Horácio C. Neto[2], João M. P. Cardoso[3]

[1]UAlg/ISE, Campus da Penha - Faro, Portugal, rmarcel@ualg.pt
[2]UTL/IST/INESC-ID - Lisboa, Portugal, hcn@inesc-id.pt
[3]*Departamento de Engenharia Informática*
Faculdade de Engenharia da Universidade do Porto (FEUP),
Porto, Portugal, jmpc@acm.org

***Abstract*- Sorting is an important operation for many embedded computing systems. Since sorting large datasets may slowdown the overall execution, schemes to speedup sorting operations are needed. Bearing in mind the hardware acceleration of sorting, we show in this paper an analysis and comparison among three hardware sorting units: Sorting Network, Insertion Sorting, and FIFO-based Merge Sorting. We focus on embedded computing systems implemented with FPGAs, which give us the flexibility to accommodate customized hardware sorting units. We also present a hardware/software solution for sorting data sets with size larger than the size of the sorting unit. This hardware/software solution achieves 20× overall speedup over a pure software implementation of the well-known quicksort algorithm.**

I. INTRODUCTION

Sorting is referred as one of the fundamental operations in computing, e.g., in a database management system the performance of the sorting algorithm is crucial for the overall performance of the system [1]. The importance of sorting is also reflected in benchmarking, in [2] a sorting based benchmark is proposed for evaluating the energy efficiency of a wide range of computer systems from clusters to handhelds. We can find sort operations in a myriad of applications. In signal processing systems, the sorting operation is used for implementing median filters [3], for Huffman coding where the Huffman tree is calculated using sort [4], etc. Nowadays, data acquisition systems with huge quantities of data to analyze are common, some of these acquisition systems the extraction of some features are performed using the sort operation [5].

Our goal is to research efficient dedicated hardware sorting units to couple to a general purpose processor (GPP) for FPGA-based embedded systems, as is illustrated in Fig. 1. In typical embedded computing systems, the data to sort are stored in a memory connected through the system bus. This memory can be located inside the FPGA in BRAMs or outside the FPGA, e.g., in an SDRAM. In this paper we present a study of three sorting units, specially designed to take advantage of the FPGA hardware resources. We compare the performance of the sorting units and their suitability to be part of embedded systems where data to be sorted are located in a single memory (typically a single-port memory). In this case, each memory access is typically only able to load/store a single data element. This usual real constraint makes sorting units that consider simultaneously access to several data elements less interesting.

The rest of the paper is organized as follows. Section 2 describes the three sorting units. Section 3 presents a hybrid software/hardware solution. Section 4 shows experimental results. In Section 5, we describe the related work, and we draw some conclusions in Section 6.

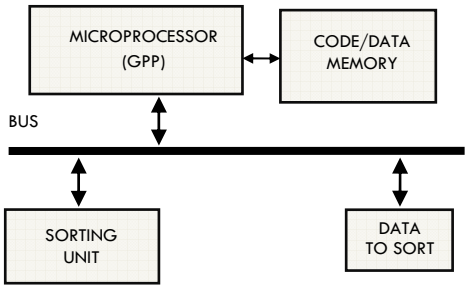


Fig. 1. Block diagram of the target system

II. HARDWARE SORTING UNITS

The three approaches for hardware sorting units, partially introduced in [6] and evaluated herein are:

- *Sorting Network Unit*: this solution is based on sorting networks where we reduce the hardware resources used for implementing sorting network by using an iterative scheme.
- *Insertion Sorting Unit*: this solution is based on a scalable and linear array.
- *FIFO-based Merge Sorting Unit*: this solution is based on the available and efficient FPGAs FIFO support using BRAMs.

Next, we describe in detail each one of the sorting units mentioned above. For the sorting units we use N to represent the total numbers of data elements to sort and M the size of the sorting machine. In the case of pure hardware solutions, such as the ones described in this section, N must be less or equal to M since a sorting machine of size M is unable to sort datasets larger than its size without a third party intervention (as is the case of the software component in the hybrid sorting unit described in section III).

978-1-4244-4649-0/09/$25.00 ©2009 IEEE

A. Sorting Network Unit

Hardware solutions using sorting networks, such as the one proposed in [7], require a large number of hardware resources to implement the complete network. They use a number of a basic comparator-swap given by $Nlog^2(N)$, being N the number of data elements to sort that in this case must be less or equal to the size of the sorting unit M. To save hardware resources, we propose an iterative sorting unit, where the sorting network is reduced to a single row (folded), i.e., to only one stage in the sorting network algorithm [6]. Fig. 2 depicts the block diagram of the Sorting Network unit. With this solution, the hardware is reused to implement all the required computing stages of the sorting network using in the limit only a single physical stage. In this case, one only needs M comparator-swap components and the addition of a simple switch network. This sorting network has the advantage of a simple control unit, and it is simple, regular and scalable.

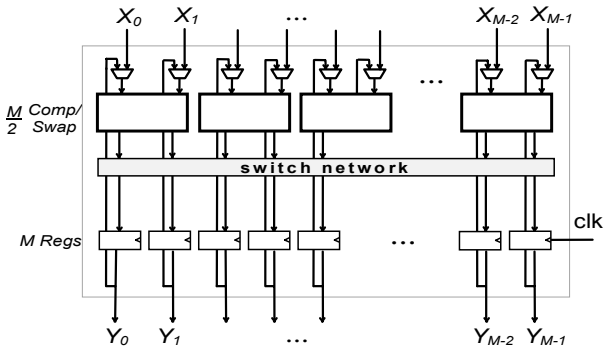


Fig. 2. Sorting Network with only one level.

The sorting network with one pipeline level refers to an implementation employing hardware reuse in every clock cycle. A switch network, implemented by an array of multiplexers, is included between the comparators and the output registers. The switch network is responsible for the data alignment, then the output is fed to the input of the unit and this loop is continuously repeated until the data input items are sorted. Until all the elements become sorted, pairs of elements are switched every clock cycle. The sorting is finished when no swap is performed in two consecutive clock cycles of the machine, considering all comparator-swap blocks, or when it reaches the final number of iterations. The computational time complexity of this sorting unit is $\mathcal{O}(N)$.

B. Insertion Sorting Unit

The Insertion Sorting unit is shown in Fig. 3 and can be seen as an array of comparison/insert cells. Each cell is composed by a comparator, a multiplexer, a register to hold data, and control logic. The array is composed of a number of these cells, corresponding to the number of elements to be sorted N, that must be less or equal to the size of the sorting unit M. The control tags work in a pipeline fashion interconnecting the cells. These tags drive the control logic located in the cell, in order to define the exact cell where this new data element is inserted.

A new data element to be sorted/inserted is broadcasted to all cells and comparisons are performed in order to find the right cell for inserting this new element. Depending on the sort direction, ascending or descending, the most right cell reflects the minimum or the maximum value. The data are read from the machine through the right cell in a sequential way (one by one), or in a parallel way, directly from each cell. In this machine, the sorting operation is overlapped with the loading of the input data. The computational time complexity of this sorting unit is $\mathcal{O}(N)$.

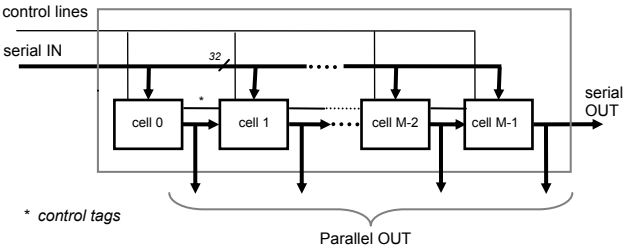


Fig. 3. Insertion sorting unit

C. FIFO Based Merge Sorting Unit

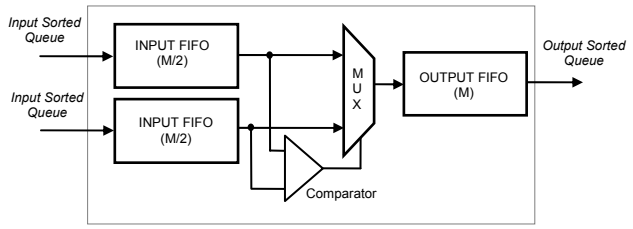
The FIFO-based Merge Sorting unit uses the merge scheme shown in Fig. 4. This unit assumes that the data in the input FIFOs have been already sorted. This first sorting stage, a primary computation (referred herein as primary sorter), can be done by the host microprocessor (e.g., using quicksort) or by another dedicated hardware sorting unit coupled to the FIFO-based Merge Sorting Unit.

The basic sorting structure illustrated in Fig. 4(a) consists of two input FIFO queues. The merging process is performed by presenting the data of the two FIFOs to the inputs of a comparator and a multiplexer. The comparator output defines which element is "greater" and signals the multiplexer control line in order to select the appropriate element to be written to the output FIFO. A new data element is sorted every clock cycle and the process repeats until all the data are processed. Both input FIFOs have size $M/2$, and the output FIFO size M, which defines the sorting unit size. This unit requires a primary sorter of size $M/2$. The computational time complexity of this approach is $\mathcal{O}(N)$.

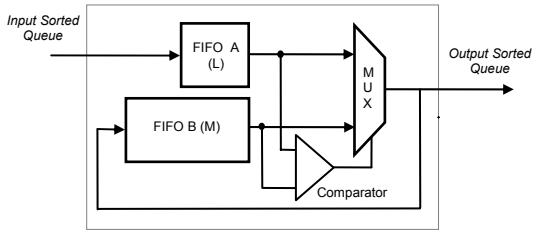
A second architecture for the FIFO-based Merge Sorting Unit uses an unbalanced FIFO merger, as illustrated in the block diagram show in Fig. 4(b). This new unit uses two FIFOs (A and B). The FIFO A is the input FIFO where the data are written in the sorting unit and has length L. The FIFO B has length M and is responsible to accumulate sorted data. This sorting unit is able to sort M values, which is considered the size of the sorting unit. This unit requires a primary sorter of size L.

The sorting operation starts with both FIFOs empty. After the first chunk of data is written in the FIFO A, a merge process is performed by presenting the output data of the two

FIFOs to the comparator and multiplexer components. Depending on the kind of sort, ascending or descending, the comparator controls the multiplexer furnishing appropriate selection signals. The first time the merge operation is performed (first iteration of the sorting unit) data are simply moved from FIFO A to FIFO B. On the other iterations, the step sorts and merges the data from the two FIFOs into FIFO B. The outputs of the two FIFOs are compared and one of the outputs is guided to FIFO B using the multiplexer. This process repeats until the FIFO B is full or the FIFO A is empty. The computational time complexity of this approach is $\mathcal{O}(N^2)$.



(a) Balanced FIFO Merger



(b) Unbalanced FIFO Merger

Fig. 4. FIFO-Based Merge sorting.

D. Summary

TABLE I summarizes the features of the three sorting units previously described. These features are related to the data read/written, hardware resources, and time complexity. The three units also feature a high degree of scalability. Thus, as long as we have hardware resources, any number of hardware elements can be added, to increase the size of the sorting units.

Sorting Networks are suitable for systems with parallel data accesses, i.e., when several data elements can be loaded and stored simultaneously. Due to the intensive use of comparator/swap components, these units require relative high hardware resources. The folded version presented in this paper diminishes the number of comparator/swap components needed and depending on the number of input/outputs saves a significant number of hardware resources.

Insertion sorting units require serial data loading and serial/parallel data storing. Its main advantage is that the sorting time is overlapped by the load time. This sorting unit can be suitable for sorting data stored on a single memory. The main drawback is the fact that it requires a number of comparators equal to the size of the sorting unit.

TABLE I
SORTING UNITS COMPARISON

Sorting Unit	Data Access		Main hardware component	FPGA hardware cost[1]	Time complexity[2]
	Input	Output			
Folded Sorting Network	parallel	parallel	comparator /swap	high	$\mathcal{O}(N)$
Insertion Sorting	serial	serial/ parallel	comparator /insertion	high	$\mathcal{O}(N)$
balanced FIFO Merger	serial	serial	FIFO	low	$\mathcal{O}(N)$
unbalanced FIFO Merger	serial	serial	FIFO	low	$\mathcal{O}(N^2)$

[1] Only the sorting unit itself. The hardware interface for read/write is not considered
2 considering sorting units of M size and data sets of N size, being N less or equal M.

The balanced and unbalanced FIFO-based merge sorting units are similar. In the balanced unit, the input FIFOs have equal size and the total size of the sorting unit is twice the size of the input FIFO. The unbalanced sorting unit requires lower size FIFOs and thus can implement bigger sorting size units, which is the main advantage over the balanced unit. Both FIFO-based merge sorting units need pre-sorting of the input data. Mainly, the hardware resources needed by these sorting units are FIFOs which can be low expensive resources in FPGAs with rich internal memories that can be customized to implement FIFOs.

III. HYBRID SOFTWARE/HARDWARE SOLUTION

When the number of data elements, N, to be sorted is larger than the size of the sorting unit, M, a sorting algorithm based on the splitting of data in chunks and on sorting operation on those chunks must be used. At the moment, we use a hardware/software implementation, based on the odd-even merge algorithm proposed by Batcher [8][9], that delegates to the hardware sorting unit the sorting of each data chunk.

The data are split into blocks (chunks) of M elements. In the first pass N/M merges are performed through the data. Note that the primary sort operation needed by the FIFO-based merge sorting unit is no more need after this first pass. The sorted blocks of data are merged in a sequential way, doing several passes through the data until the whole data elements become sorted, the total number of merges required is given by:

$$\#Merges = \frac{N}{2M} log\left(\frac{2N}{M}\right) log\left(\frac{N}{M}\right) + \frac{2N}{M} - 1 \quad (1)$$

Since we are talking about a sequential sorting system, all the components in the chain have an impact on the global system performance. We have also to consider the software overhead due to the Odd-Even merger implementation, $SW_{Overhead}$. The global sorting time can be represented by:

$$T(N) = \left(\frac{N}{M} + 2 \cdot \#Merges\right) \cdot T_{com.}(M) + \left(\frac{N}{M}\right) \cdot \left(\frac{M}{L}\right) \cdot T_{Primary}(L) + \\ + (\#Merges) \cdot T_{FIFO}(M, L) + SW_{Overhead} \quad (2)$$

where $T_{Com.}(M)$ represents the total time to communicate M elements between memory and the sorting unit. This value remains constant for different L values. $T_{FIFO}(M, L)$ represents the time spent by the sorting unit to sort M elements, with an input FIFO with size L. Finally, $T_{Primary}(L)$ represents the time needed by the primary sort to sort L data elements before transferring them to the second component of the sorting unit.

In this analytical model we consider an implementation of the FIFO-based sorting unit that merges one data element every two clock cycles, one cycle is for reading from the FIFO and the other cycle is for writing to the FIFO.

IV. Experimental Results

The sorting units (including their interface and control units) have been specified in parameterized behavioral RTL-VHDL code. An FPGA development board, consisting of a Xilinx Virtex 5 SXT FPGA (XC5VSX50T-1ff1136), SDRAM, and other peripherals, has been used to characterize the FPGA implementation of the units and to implement and test the overall system.

The Embedded Development Kit (EDK) and ISE, release 10.1iSP3, from Xilinx were used for system development, configuration, RTL synthesis, and placement and routing. For prototyping and test, we use an embedded system with a Xilinx 32-bit MicroBlaze *softcore* processor as GPP. The MicroBlaze was configured with default parameters, i.e., without the optional datapath units, and without caches. The software implementations were compiled using the C compiler included in the EDK (mb-gcc), with the O2 option selected.

For the particular system presented herein, the sorting units are instantiated as a Processor Local Bus (PLB) custom core and the data to be sorted are stored in the 256 MB DDR2 SDRAM connected to the PBL bus.

At first we implemented and evaluated a pure software sorting using this scheme with the quicksort algorithm.

Our analysis is performed in two steps, one regarding the FPGA resources and the other the execution time of the sorting units being evaluated. The results obtained by the proposed sorting unit are compared with a software implementation of the quicksort algorithm. In these experiments, both the sorting unit and the *softcore* processor run at the same clock frequency (100 MHz for the validations done using the FPGA board). For decreasing the time spent in communications, we also consider DMA access between the sorting unit and the main memory.

To implement the FIFOs in the target FPGA, we use the Xilinx CORE Generator [9].

TABLE II summarizes the FPGA resources used for the sorting units for datasets consisting of 32-bit integers. As can be seen, the sorting network implementation, even with only one pipeline level and size of 128 elements, takes about 50% of the overall FPGA resources, For the balanced FIFO-based merge sorting unit the hardware resources are residual: less than 1%, for a size of 1k. For the unbalanced 32 k FIFO-based merge sorting unit only 22% of the BRAMs are used.

The maximum clock frequencies, reported by the Xilinx ISE tool, are shown in Table II. They range from 156 to 265 MHz, which exceeds the current maximum PLB frequency that can be achieved (150 MHz in the Xilinx Virtex 5 devices). Thus, the maximum operating frequency for the sorting units is not a limitation for our target evaluation system.

With respect to maximum operating clock frequencies, the fastest sorting unit is the Insertion Sort. However, this sorting unit consumes many hardware resources that prevents it to be used for large M sizes (a 128, 32-bit Insertion Sorting unit requires 50% of the LUTs of the FPGA being used). Compared to the Insertion Sorting unit, the Sorting Network unit is a parallel sorting unit that is able to sort very fast the number of elements available in its inputs. This can be an interesting property when the application or the target architectures has the data to be sorted distributed in a number of on-chip memories/registers.

The FIFO-based Sorting units are able to use efficiently the on-chip memories (BRAMs), customized as FIFOs, and make possible the implementation of large hardware sorting units. In this case, a 32 k 32-bit sorting unit uses 22% of the BRAM FPGA resources and about 1% of the FPGA logic resources. These units are, however, the ones with lower maximum operating clock frequencies (156 and 166 MHz).

TABLE II
Device Resources Utilization Considering a XILINX XC5VSX50T FPGA

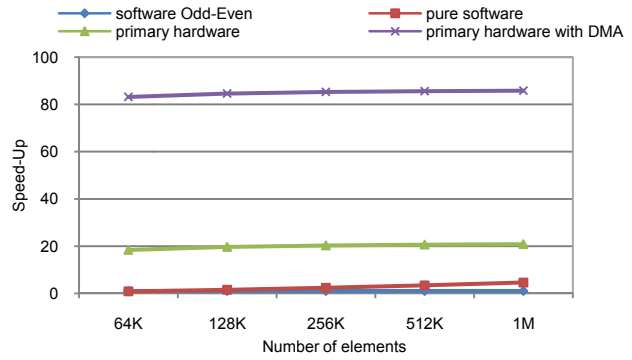
Sorting Unit	Unit Size[1] (×32 bit)	#Slice Registers[2]	#Slice LUTs[2]	#BRAM[2]	Max. Frequency (MHz)
Folded Sorting Network	128	4387 (13%)	14923 (45%)	0	194
Insertion Sorting	128	4223 (12%)	16420 (50%)	0	265
Balanced FIFO Merger	1 k	438 (1%)	362 (1%)	3 (2%)	156
Unbalanced FIFO Merger	32 k	559 (2%)	487 (1%)	30 (22%)	166

[1] 32-bit width data elements.
[2] Percentage values refer to FPGA utilization

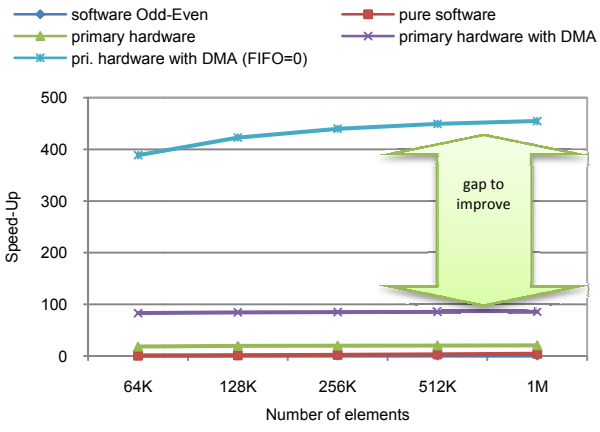
We now analyze the performance achieved by the sorting units when considering an hardware/software solution. This hardware/software sorting unit was implemented as a hybrid approach with an Insertion sorting unit as primary sorting component and an unbalanced FIFO-based Merge sorting unit, using for FIFO A and B, 1 k words (L) and 32 k words (M), respectively, as secondary sorting component.

The results presented in Fig. 5 include:

- a software/hardware solution with a software even-odd sorting algorithm, a software primary sorter and the FIFO-based Merge Sorting Unit with 32 k words as its size;
- a software/hardware solution with a hardware primary sorter (size 1 k words) and the FIFO-based Merge Sorting Unit with size 32 k words with and without DMA;
- a pure software implementation of the quicksort algorithm;
- a pure software implementation using an even-odd sorting algorithm using quicksort to sort groups of 32 k words;
- an hypothetical and theoretical case considering that the total time spent in the hardware sorting units is zero (i.e., considering only the communication and odd-even software execution time). The gap between the performances of the fastest real solution presented in this paper and this theoretical performance is presented in the Fig. 5. By showing the upper limit of the hybrid sorting solution, this gap shows us the maximum improvements we might achieve with fastest sorting units (if available).



(a)



(b)

Fig. 5. Performance results for sorting large data sets and using a hw/sw solution: (a) Evaluation of the hardware/software sorting system for sets of data with sizes from 64 k to 1 *M* 32-bit integers. A hybrid sorting unit is considered using insertion sorting as primary hardware for the unbalanced 32 k FIFO. A comparison is done between the unit with and without DMA communication, with Merge software and pure software quicksort; (b) Gap to improve with a theoretical upper bound referring an hypothetical hardware/software solution with an hardware sorting unit with zero sorting time.

We obtain a high speed-up, about 20×, when compared to a pure software quicksort solution, running in the microprocessor of the system, and about 80×, when compared to the software odd-even merge implementation with sorting done by a pure software solution. Note that in the software/hardware solution, the primary sort is only needed in the first pass through the overall data.

V. Related Work

Concerning application-specific architectures, two different approaches have been mainly considered for accelerating sorting operations, one focusing on variations of the sorting networks [7][8][11], suitable for data available in a parallel fashion and the other exploring systolic linear arrays [12][13], suitable for sequential data input. Although those approaches may achieve high-performance sorting operations, both rely on the simultaneous availability of data to feed the sorting units. For most cases, this hampers their practical use with current technology or at least in systems where the data to be sorted are all located in the same memory device (usually with a single port).

Ratnayke and Amer [14] propose an FPGA implementation variation of the counting sort algorithm [15]. The occurrence histogram is implemented with on-chip FPGA memories (BRAMs). This approach is efficient, but only for a relative small data ranges, typically less than 4K elements.

Due to the importance of sorting operations, parallel processing sorting solutions have also been [16][17][18], these solutions are suitable for the multicore architectures and can be complemented by hardware sorting units, as the ones analyzed in this paper.

Our approach mainly distinguishes from the previous hardware based approaches by tackling the problem considering the acceleration of sorting operations when dealing to systems where data to sort are typically stored in a single port memory and dealing with large datasets. Our approach might also be used to speedup parallel implementations of the sort operation.

VI. Conclusions

Sorting is becoming an important operation for many embedded computing systems. However, sorting large datasets is a computationally demanding operation. Thus, speeding-up the sorting operation is very important for many computing systems.

In this paper we evaluated three hardware sorting units and proposed a hybrid software/hardware sorting solution. All the sorting units have been implemented on an FPGA board mainly consisting of a Virtex5 FPGA and external memory. Then, their performance has been evaluated by measuring the execution time of both individual hardware solutions and hybrid hardware/software solutions.

The sorting units focus on three different architectures: single stage sorting networks (folded), an insertion sorting array, and two FIFO-based merge sorting units. When sorting large datasets is needed, and since the total number of data

elements is greater than the size of the sorting units, an odd-even merge scheme controlled by software is proposed. This sorting solution is a hybrid approach with a primary hardware Insertion Sorting unit and an unbalanced FIFO-based Merge Sorting unit. This hybrid sorting unit provides speed-ups of about 20×, when compared to a quicksort pure software solution running in the microprocessor of the system, and about 80× when compared to the pure software odd-even merge implementation.

The results presented show that a hybrid sorting solution can be used to speedup sorting operations over large datasets. These obtained speedups might also be important to save energy since they may permit the slowdown of the clock frequency, maintaining the original performance.

ACKNOWLEDGMENTS

This research was partially supported by FCT, the Portuguese Science Foundation, under the research grant PTDC/EEA-ELC/70272/2006.

REFERENCES

[1] G. Goetz, "Implementing sorting in database systems," *ACM Comput. Surv.*, vol. 38, pp. 10, 2006.

[2] R. Suzanne, A. Mehul, Ranganathan Parthasarathy and Kozyrakis Christos, "JouleSort: a balanced energy-efficiency benchmark", *2007 ACM SIGMOD international conference on Management of data,* 2007

[3] S. Lukáš, "Evolutionary Design Space Exploration for Median Circuits", *Lecture Notes in Computer Science,* Vol. 2004, No. 3005, DE, pp. 240-249, 2004.

[4] E. Jamro, M. Wielgosz, and K. Wiatr, "FPGA Implementation of the Dynamic Huffman Encoder," *Proc. Workshop of Programmable Devices and Embedded Systems,* pages 60-65, February 2006.

[5] C. C. W. Robson and C. Bohm, "A high speed data acquisition collector for merging and sorting data," *Nuclear Science Symposium Conference Record,* 2008. NSS '08, 2008.

[6] R. Marcelino, H. Neto, and J. M. P. Cardoso, "Sorting Units for FPGA-Based Embedded Systems," *IFIP International Federation for Information Processing, Volume 271*; Distributed Embedded Systems: Design, Middleware and Resources; Bernd Kleinjohann, Lisa Kleinjohann, Wayne Wolf., pp. 11–22., 2008.

[7] Y. Zhang and S. Zheng, "An Efficient Parallel VLSI Sorting Architecture", *VLSI Design*, vol. 11, no. 2, pp. 137-147, 2000.

[8] K. E. Batcher, "Sorting networks and their applications," *Proceedings of the April 30--May 2, 1968, spring joint computer conference*. Atlantic City, New Jersey: ACM, 1968.

[9] S. Robert, Algorithms in C: Parts 1-4, *Fundamentals, Data Structures, Sorting, and Searching*: Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] Xilinx Inc. "Virtex-5 FPGA User Guide. UG190 (v4.5)" January 9, 2009.

[11] J. Martinez, R. Cumplido and C. Feregrino, "An FPGA-based parallel sorting architecture for the Burrows Wheeler transform", *International Conference on Reconfigurable Computing and FPGAs, 2005. ReConFig 2005.* , Vol. pp. 2005

[12] R. Perez-Andrade, R.Cumplido, F. Del Campo and C. Feregrino-Uribe, "A Versatile Linear Insertion Sorter Based on a FIFO Scheme", *IEEE Computer Society Annual Symposium on VLSI*, pp. 357-362, 2008

[13] B. Parhami and K. Ding-Ming, "Data-driven control scheme for linear arrays: application to a stable insertion sorter," *IEEE Transactions on Parallel and Distributed Systems,* vol. 10, pp. 23-28, 1999.

[14] K. Ratnayake and A. Amer, "An FPGA Architecture of Stable-Sorting on a Large Data Volume : Application to Video Signals", *41st Annual Conference on Information Sciences and Systems,* 2007. CISS '07., pp. 431-436, 2007

[15] H. H. Seward, "Information sorting in the application of electronic digital computers to business operation," MS thesis, Massachusetts Institute of Technology (MIT),1954.

[16] M. Edahiro, "Parallelizing fundamental algorithms such as sorting on multi-core processors for EDA acceleration," *Design Automation Conference, 2009.* ASP-DAC 2009. Asia and South Pacific, 2009.

[17] H. Inoue, T. Moriyama, H. Komatsu and T. Nakatani, "AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors", *16th International Conference on Parallel Architecture and Compilation Techniques*, 2007. PACT 2007 , pp. 189-198, 2007

[18] E. Herruzo, G. Ruiz, J. Ignacio Benavides and O. Plata, "A New Parallel Sorting Algorithm based on Odd-Even Mergesort", *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing,* 2007. PDP '07, pp. 18-22, 2007