On Implementing Sorting Network Machines with FPGAs

Rui Marcelino
UALG/EST - Faro
rmarcel@ualg.pt

Horácio Neto
IST/INESC-ID
hcn@inesc-id.pt

João M. P. Cardoso
IST/INESC-ID
jmpc@acm.org

Abstract

This paper presents two implementations of sorting network machines, bearing in mind embedded computing systems. In the proposed implementations, the hardware is reused by iteratively performing the sorting algorithm. The two implementations have been tested using Field Programmable Gate Arrays (FPGAs). Experimental results show the differences in performance, area, and power dissipation between the two implementations.

1. Introduction

Search and sorting are becoming important features for embedded applications. Colton [1] refers that the need for search is driven by both technical and social factors. On the technical side, the plummeting price of both Flash and micro-drive disks means that even modest devices are being equipped with amounts of storage that were unthinkable only a couple of years ago. On the social side, Google has conclusively demonstrated the value of web-based search, and users are now expecting Google-like features on their PDAs, cell phones and other devices.

Sorting is an integral component of most database and data stream management systems. The performance of queries in these systems is often dominated by the cost of the sorting algorithm. Hence, sorting machines able to improve performance are required.

Sorting has been exhaustively studied in the area of computer science and many sorting algorithms exist [2]. On a von Neumann/Harvard architecture with sequential instruction execution, Quicksort is the fastest of the common sorting algorithms for general case sorting [2] . However, sorting remains a time spending operation. The use of Hyper-Threaded technology to optimize Quicksort in the Intel compiler [6], the use of graphics processor [7], and the use of dedicated architectures [3][4][5], are three examples of efforts to accelerate sorting operations.

This paper is organized as follows. Section 2 defines and explains parallel sorting networks and the parallel odd-even transposition algorithm, Section 3 describes the two sorting machines implemented in this work. Section 4 gives some experimental results and section 5 gives some conclusions and future work.

2. Background

In this work we are interested to research sorting machines based on a *comparison-swap* network model of computation [8], in which several comparisons are simultaneously performed in order to speed-up sorting.

A sorting network (see Figure 1) is a circuit that receives n inputs, x_0, x_1, x_2, …, x_{n-1} and permutes them to produce n outputs y_0, y_1, y_2, …, y_{n-1} such that the outputs satisfy $y_0 \leq y_1 \leq y_2 \leq \ldots \leq y_{n-1}$.

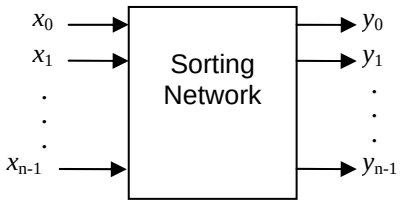

Figure 1. Sorting network representation

Figure 2 shows a graphical representation of a sorting network. A horizontal line represents each input of the sorting network and a connection between two lines represents a *comparator-swap*, which compares the two elements and exchanges them if the one on the upper line is larger than the one on the lower line. Elements at the output are sorted, with the largest element on the bottom line.

A sorting network can be classified by two different parameters:

- Its *depth*, which is defined as the number of parallel steps that it takes to sort any input, given that, in one step, disjoint comparison-swap operations can take place simultaneously. This is a time parameter, usually also referred to as stages' number.

- Its *length*, or size, which is the number of total comparison-swap used in the sorting network. This is a space parameter, usually also referred to as comparators' number.

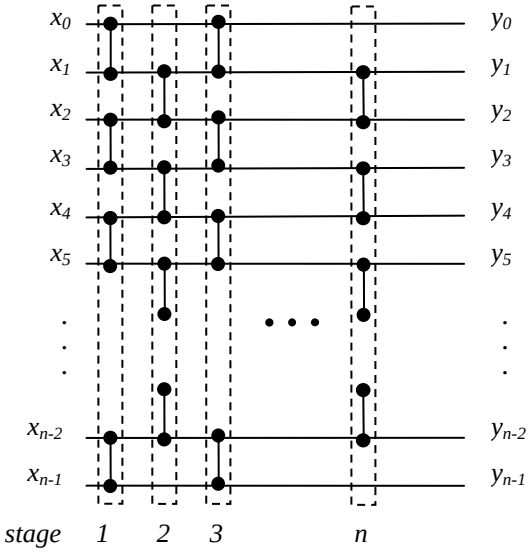

Figure 2. Graphical representation of the Odd-Even transposition sorting network algorithm

The odd-even transposition sorting network for *n* input data items consists of *n* comparators stages [2], which means that have a *n* stages *depth*. In each stage, either all inputs at odd index positions, or all inputs at even index positions, are compared with their neighbors. Odd and even stages alternate as is shown in Figure 2. The total number of comparators is equal to $n \cdot (n\text{-}1)/2$. The odd-even transposition sorting network is not the algorithm that performs the quickest sort, but has the advantages of simplicity, locality and scalability. Locality means that comparators exist only between adjacent elements; scalability means that the structure can be easily extended for larger sizes. These features are the keys for the two hardware implementations presented in this paper.

3. Odd-Even Sorting Network Machines

The basic element of sorting networks is the comparator-swap block which performs the elementary sort between two elements. The comparator-swap block is shown in Figure 3

The inputs of this block are the two data elements to be sorted A, B and the clock signal. The outputs are L and H with the two sorted data elements, with L less then or equal a H and a CHANGE signal that flags if a swap between the two input data elements has been done.

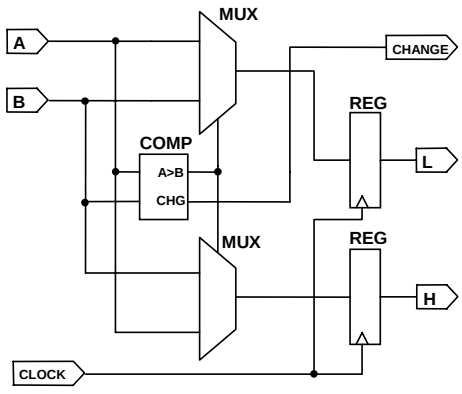

Figure 3. Comparator-Swap block

Since the hardware implementation of the sorting network, represented in Figure 2, may require too many FPGA resources, especially when dealing with a large number of inputs, a split of the network in sequential stages is evaluated. On this implementation, referred herein as "sequential network", the hardware is re-used to implement all the required computing stages of the sorting network by a short number of physical stages. Two schemes have been implemented using the odd-even transposition sort algorithm. Note that this sorting network is used for its simplicity, regularity and scalability. The hardware implementation is regular and requires a simple control unit.

Two implementations have been considered. The first one named sequential-I, refers to a machine employing hardware reuse in every clock cycle. For this, a basic comparator-swap is used as showed at figure 3, but without output registers. Registers are placed in the end of the stage to store the results every clock cycle (see Figure 4). A switch network is included between the comparators and the output registers. The switch network is responsible for the data alignment, as show in Figure 5. Data to be sorted are input to the sorting machine. Then, the output data is feed to the input of the machine and this loop is continuously performed until data items are sorted. Since on each stage the inputs of the comparators (except on the first sorting iteration) are the outputs from the registers, an alignment must be done between the outputs from the comparator-swap blocks and the register inputs. This alignment is done by an array of multiplexers that implement a switch network. Until the end of the sorting, an

alternate data switching is performed on every clock cycle. The sorting is finished when no swap is performed in two consecutives clock cycles of the machine, considering all comparator-swap blocks or when reaches the final number of iterations *n*. This is done by the control logic that reads the output global flag, CHANGE, which is an AND of all the individual CHANGE flags.

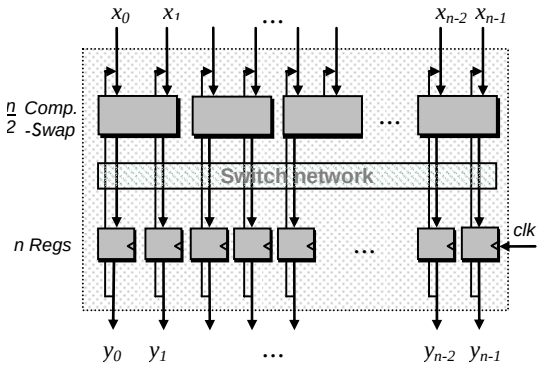

Figure 4. Sequential-I, sorting machine

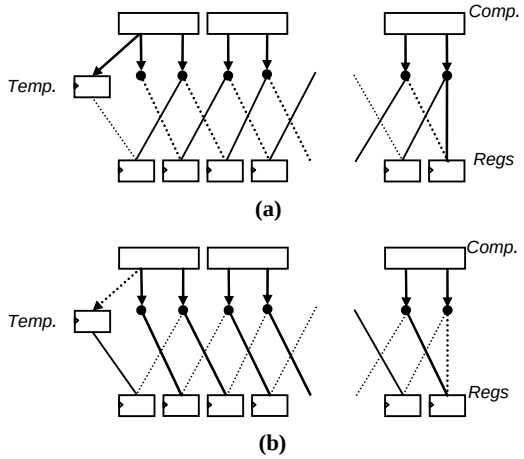

Figure 5. Switch network for data alignment on sequential-I (a) Odd; (b) Even

The other sequential approach implemented is referred to as sequential-II. The algorithm used is the same, the odd-even transposition sorting network, but the data alignment is performed by the use of two comparator levels (see Figure 6). Now the comparator-swap blocks have the outputs registered, creating a 2-stage pipelined machine (see Figure 3). The hardware is reused every two clock cycles. As before, the sorting finishes when the control logic detects that no swap was performed on all comparator-swap blocks. This machine uses two clock cycles more for the pipeline prologue.

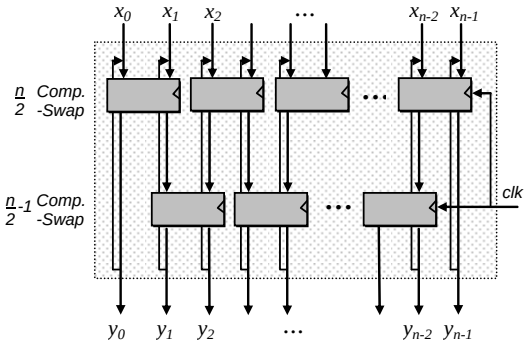

Figure 6. Sequential-II, sorting machine

4. Experimental Results

The two sorting machines and their control units have been specified in VHDL. A Spartan 3 and a Virtex 4FX FPGAs from Xilinx have been used to characterize the FPGA implementation of the two machines. These particular sorting machines work with 16 elements of 8-bit data width.

The Xilinx ISE 8.2i [9], tool is used for synthesis and place and route. The estimated results reported by the ISE tool after P&R are presented in Table I and Table II. The last column of the tables shows an estimation of the average power dissipated for implementations using the maximum clock frequency.

Machine	#4-LUT	#FFs	#slices	Freq. (MHz)	Power (mW)
Sequential-I	601	139	320	90	59
Sequential-II	703	243	383	160	62

Table I. Results with an FPGA SPARTAN-3 (x3s400-5fg456)

Machine	#4-LUT	#FFs	#slices	Freq. (MHz)	Power (mW)
Sequential-I	609	139	320	172	62
Sequential-II	699	243	381	350	154

Table II. Results with an FPGA VIRTEX-4 FX (xcvfx12-12ff668)

Comparing the two implementations (see Table I and Table II), the Sequential-I requires fewer FPGA resources than the Sequential-II machine. The later, requires 20% more hardware, but the maximum clock frequency increases about 80% and 100%. This means that with the sequential-II a higher throughput of sorted elements can be achieved. Note, however, that this higher frequency implied a 2.5x more power dissipated using the Virtex-4 FPGA. Surprisingly, the power dissipated

remains practically the same for the Spartan 3 FPGA (Table I).

Table III illustrates the latency for the machines considering sorting networks of *n*-input operands all available at the same time (i.e., the latency of the sorting machine without taking into account how input/output data are loaded/stored). For maximum latency sequential-II requires two additional clock cycle to execute, as has been explained before.

Machine	Latency (# clock cycles) to sort N-input elements	
	minimum	maximum
Sequential-I	2	N
Sequential-II	2	N+2

Table III. Latency for the proposed sorting machines when considering an N-input sorting network

5. Conclusions and Future Work

In this paper we have derived two sorting machines to implement the odd-even transposition sorting network. These two machines implement the sorting network by sequencing stages of the network on a physically available single hardware stage. This can be an important feature to save FPGA resources without reducing throughput. Evaluations of two examples of those machines, using 16 8-bit inputs and Xilinx Spartan-3 and Virtex-4 FPGAs, have been included.

Ongoing work is addressing the implementation of a sorting network machine without using the reuse of stages proposed in this paper. This will be used to compare that implementation with the ones presented in this paper. Future work will consider the implementation of other sorting networks and the integration of the sorting machines with a microprocessor core to evaluate software and software/hardware solutions.

References

[1] M. Colton, "How to Add Relational Database Search to Your Embedded Device application", Dr.Dobb's Journal, November 2006.

[2] *D.E. Knuth, The Art of Computer Programming*, Vol. 3 - Sorting and Searching". Addison-Wesley 1973.

[3] C. S. Lin, B. D. Liu, "Design of a Pipelined and Expansable sorting Architecture with Simple Control Scheme", IEEE International Symposium on Circuits and Systems, 2002

[4] J. Harkins, T. E. Ghazawi, E. E. Araby, M. Huang, "Performance of Sorting Algorithms on the SRC 6 Reconfigurable Computer", Proceedings IEEE International Conference on Field-Programmable Technology, 2005

[5] J. Martínez, R. Cumplido R., C. Feregrino, "An FPGA-based Sorting Architecture for the Burrows Wheeler transform", Proc. of the 2005 International Conference on Reconfigurable Computing and FPGAs, 2005.

[6] R. D. P. Rajiv, "Accelerating QuickSort on the Intel® Pentium ® 4 Processor with Hyper-Threading Technology", http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/threading/hyperthreading/20372.htm?page=1, March 2003.

[7] N. Govindaraju, N. Raghuvanshi, M. Henson, D. Tuft, D. Manocha, "A Cache-Efficient Sorting Algorithm for Database and Data Mining Computations using Graphics Processors", UNC Technical Report 2005.

[8] K. Batcher, "Sorting Networks and Their Applications" Proc. AFIPS Spring Joint Computer Conf. Vol.32, pp. 307-314, 1968.

[9] Xilinx: The Programmable Logic Company, http://www.xilinx.com/