HARDWARE ACCELERATION OF SORTING ALGORITHMS USING RECONFIGURATION TECHNICS

Paweł Russek, Kazimierz Wiatr

AGH University of Science and Technology
Department of Electronics
al. Mickiewicza 30
Cracow, POLAND

Abstract: This paper presents authors' research on implementation of sorting nets in Field Programmable Logic Gates. As a theoretical base bitonic sorting nets were considered. During their research authors met several difficulties which came from big digital resource requirements necessary to realize medium sizes sorting nets. A few techniques are proposed to obey these difficulties and advantages of sorting nets implementation in reprogrammable hardware are emphasized.

Keywords: algorithms, concurrent architectures, computing systems, computational methods, parallel algorithms.

1. INTRODUCTION

Sorting operation is one of the most common and probably most often performed operation. It is crucial part of many algorithms realized by data computing machines. Sorting is particularly important in data collection solutions i.e. database systems where several indexing techniques allow fast searching and rapid access to saved data. To build any index system sorting algorithms are required. At present when huge number of data are processed and stored (Internet servers for example) efficient sorting seems to gain bigger importance than ever. Especially when so called search engines are considered. It is also the truth that the main applications of corporate servers are data base systems. Maintains of them is the main computational job of the digital machines where databases are installed. Database queries service also needs sorting for successful completion of queries each time they are sent to the server. These facts are not meaningless to the servers' processors because sorting is time consuming and exhaustive operation.

There are many sorting algorithms designed. Their important feature is computational complexity (Knuth D. E , 1998). To start with the simplest bubble sorting or sorting through insertion (both with complexity $O(n^2)$) and finish with most powerful quick sort algorithm (with average complexity $O(nlgn)$). These are sequential algorithms working according paradigm common to broad range of present computational machines. There are also others sorting methods which regularity allows implementation based on parallel and pipeline techniques (Flynn M. J., 1966). Such processing introduce completely new opportunities when building computing systems. Of course this kind of data processing is not odd to the currently exploited processors and computer architectures. For example modern processors feature is pipelining in processing program instructions (Hwang K. 1993). Also commonly meet DSP algorithms can be performed by special purpose co-processors which dedicated architectures exploit possibility of instruction execution in parallel and/or serial manner. Parallel computing occurs on different levels of data processing. There should be distinguished different situations: when sequential algorithm is executed by pipeline architecture processor (for example DSP processor), when algorithm is parallely performed by special purpose dedicated architecture (for example video compression acceleration chip) and when data processing is performed either on multi processors platform or by PC computers cluster. The focus of this paper regards to second situation when algorithm's dedicated architecture is proposed. When sorting algorithms are considered sorting nets are pipeline and parallel executed solutions which can be hardware implemented (Cormen T.H *at al*., 2001).

The advantages of Field Programmable Gate Arrays (FPGA) in the area of hardware acceleration of computation are well known (Wiatr K., 2003). The major gain when FPGA are used as an implementation platform is that hardware structure can be easily adopted to the currently performed task. Thanks to FPGA digital gate resources can be saved because the same digital gates set can be used for different hardware structures. Structures can be changed at system work time. Similarly to the techniques DSP algorithms and architectures were successfully adopted to reprogramable structures it is also possible to build digital nets which according to the pipeline/parallel processing scheme can accelerate sorting jobs. This paper presents the FPGA implementation results of sorting nets. It is particularly focused on number of necessary digital resources to realize desired size sorting net. Several solutions are also proposed to obey the sorting net size limitations caused by number of gates available in present FPGA.

2. SORTING NETS

Sorting nets work principle based on ordering element presented on figure 1.

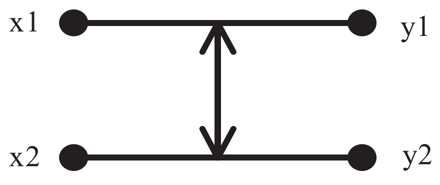

Fig. 1. Ordering element

Ordering element consist of two inputs and two outputs. Element moves data presented on its inputs to outputs in such a way that smaller value always appears on particular output. It is assumed that smaller value appears on *y1* output and bigger value appears on *y2* output in this paper. In fact such an element can be considered as sorting net of two elements lists. On the other hand it is possible to build any number of elements sorting net using appropriate ordering element structure. Figure 2 presents an example of sorting net realization for four elements lists. Sorting nets are circuits built of ordering elements. It should be highlighted that each ordering element works independent of other elements so we can call this parallel processing. Also elements' outputs are connected to other elements' inputs. If such net's layers are separated by memory elements (Flip-Flop for example) we can process data according pipeline scheme where at the same time several lists can be sorted. Of course different lists are at different sorting stages and result of sorting only one list is available at the particular moment. The advantage of memory element utilization is that subsequent lists can be faster presented to the sorting net's inputs. There is still a delay between input and output of net but overall performance is better - more lists can be sorted at the same time. If memory elements are flip-flops sorting net output is delayed *n* clock cycles with respect to the input (where *n* is number of flop flops layers between inputs and outputs of sorting net).

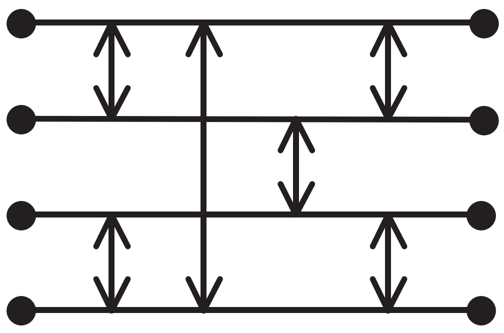

Fig. 2. Sorting net of four element lists.

3. BITONIC SORTING NETS

Bitonic sorting nets are sorting nets which are constructed according rules presented in this paragraph. The name comes from bitonic sequences. Bitonic sequence is a sequence which is not ascending first and than not descending. For example sequences <1, 3, 5, 9, 6, 2> and <9, 8, 3, 4, 7, 8> are bitonic (Cormen T.H *at al*., 2001). In the other words bitonic sequence is created by juxtaposition of two sequences: increasing and decreasing. The second definition is not meaningless for the rules the bitonic sorting nets are built. The main idea of bitonic net is presented on figure 3.

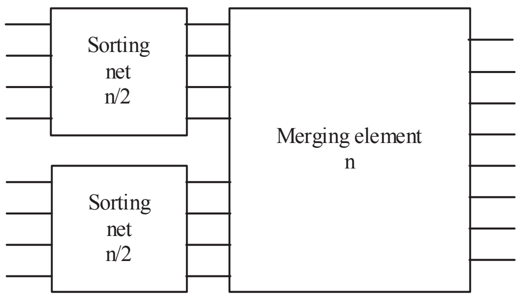

Fig. 3. Sorting net recurrence idea

As it can be seen on the picture to build n-input sorting net two $n/2$-input sorting nets and merging element are used. According to this idea to sort n-element sequence it should be first divided into two $n/2$-elment sequences, each $n/2$-element sequence is sorted separately and than two sorted sequences are merged to compose sorted n-element sequence. The same scheme can be used to build $n/2$-sorter: two $n/4$-element sorters are necessary and $n/2$-element merger. Such recurrence leads to 2-elements sorters which are in fact ordering elements. The only question is how to create merging element which is a crucial element on every stage of proposed recurrence. To explain the rules merging element is constructed the following theorem will be useful:

If a_i, $0<i<2n$ is a bitonic sequence and e_i , d_i, $0<i<n$ are sequences of a_i elements such that $e_i=max(a_i, a_{i+n})$; $d_i=min((a_i, a_{i+n})$ then e_i , d_i are also bitonic sequences and $max(d_i)<min(e_i)$.

Example.
a_i=<1,2,4,7,9,10,12,14,15,10,5,4,3,2,1,0>; 0<i<16
Construction of e_i and d_i requires comparison of the first and the second half of a_i

a_i=	<1,	2,	4,	7,	9,	10,	12,	14>
a_{n+i}=	<15,	10,	5,	4,	3,	2,	1,	0>
$e_i=max(a_i,a_{i+n})$=	<15,	10,	5,	7,	9,	10,	12,	14>
$d_i=min(a_i,a_{i+n})$=	<1,	2,	5,	4,	3,	2,	1,	0>

It can be seen in an example that e_i and d_i are in fact bitonic and every element of d_i is smaller than each element of e_i. Operation performed in theorem is graphically presented on figure 4.

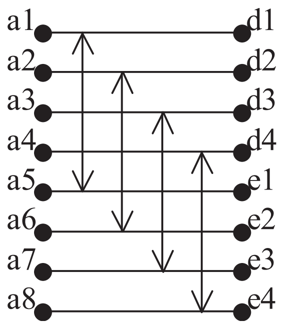

Fig. 4. Half cleaner. Primary form.

If assumed that a_i was composed as a juxtaposition of two sorted ascending sequences: m_i and n_i then operation from figure 4 can be re-expressed in a form shown on figure 5. Net presented on figure 5 is called half cleaner. Half cleaner allows to merge to sorted sequences and what is most important to create two bitonic sequences that fulfil feature $max(d_i)<min(e_i)$. Presented bitonic sequences' features leads to the recurrence scheme that allows to build merging element. It is shown on figure 6.

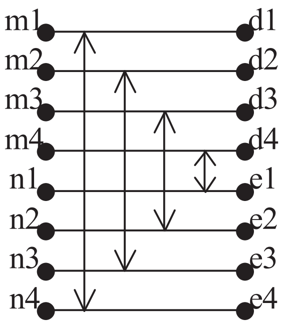

Fig. 5. Half cleaner. Secondary form.

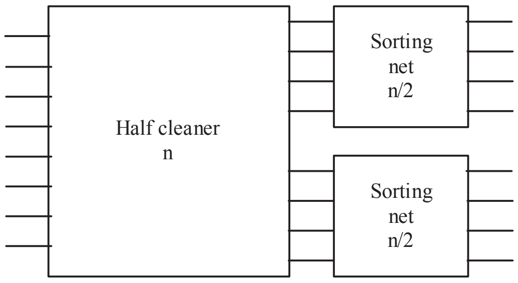

Fig. 6. Merging element recurrence formula.

4. HARDWARE IMPLEMENTATION

There are several difficulties when sorting net hardware implementation is to be realized. The first and essential problem is that to create a net with number of inputs big enough to be useful from the practical point of view a lot of resources is needed. Implemented net should have much enough input to compensate an effort to exchange easy to implement sequential algorithms with parallel hardware solution. Table 1 presents number of ordering elements necessary to implement different sizes sorting nets.

Table 1. Number of ordering elements for some sorting nets' sizes

Sorting net number of inputs	4	8	16	32	64
Number of ordering elements	6	24	80	240	672

Table 1 shows that number of necessary sorting elements increase very fast with increasing number of sorting net input. Of course cost of ordering element realization depends on chosen technology but cost of realization even medium sizes sorting nets despite the technology is huge. It should be highlighted here that cost of ordering element vary with number of single input bit width. The second problem is how to achieve a proper data interface to transfer data to implemented sorting net. Even if number of inputs is medium size and a bit width of inputs is average the number of available chip pins is easily exhausted. Above problems doesn't mean that there is no way to take an advantages of hardware acceleration of sorting tasks. The techniques to omit mentioned problems will be proposed.

Sorting co-processor. This method requires division of sorted sequence into sub sequences which length fit the size of available sorting net. These sub sequences are sorted in hardware and then sorting process is completed by software (or hardware) by merging sorted sub sequences. This merging can be done according followed algorithm: *Take the first elements from two sorted sequences The smaller element is added to the newly created sorted sequence and the bigger element comes back to its origin position Repeat this as long as one of the sequences is empty. Add remaining sequence at the end of new sequence.* This procedure can be realised by software or by hardware but in both situation it is sequential. Computational effort of such algorithm is in the worst case *O(nlg(n/k))* where k is the size of available sorting net. The cost of sorting in hardware sorting net is not included in this formula because it is assumed that it is much smaller than cost of sequential algorithm and such meaningless. In compare with fast-sort algorithm which cost is *O(nlgn)* accelleration of sorting when hardware coprocessor is used is:

$$s = \frac{1}{1 - \frac{\lg k}{\lg n}} \qquad (1)$$

where:
k - sorting net size
n - sorted sequence length

If k goes to n then s goes to infinity in formula. It is because assumption was made there is no time delay when hardware sorting is performed. The formula is not exact but the conclusion can be made that hardware acceleration speeds sorting up.

Serialization. The second method that allows to fit sorting net in limited number of logic gates is serialization of sorting element. In this solution the data is not loaded to the net parallelly but serially starting from the most significant bit. In each sorting net node (sorting element) values are compared bit by bit. The value which is first '1' when the other value is '0' is assumed to be bigger and until last bit is transferred to the bigger output. In this case memory element is necessary to save which input is bigger. This increase implementation cost of memory element but on the other hand serialization and memory element adding is the only practical way to sort long alphanumerical strings. Serialization can be utilized in many different ways and the only condition is that consecutive bits must have decreasing weight. This for example eliminate exclude U2 coded values serial sorting. In such situation binary representation has to be changed first.

5. FPGA IMPLEMENTATION

Real implementation of sorting nets in FPGA began with programming a dedicated tool able to generate sorting net with desired number of inputs and suitable inputs' bit width. Because of dynamic nature of every FPGA circuit many different sorting net structures were necessary. Like no other technology reconfigurable FPGA logic allows to accommodate sorting net structure to temporary needs. According to data structure proper sorting net is chosen from the virtual hardware library. Because it was necessary to create many implementation to fill proposed hardware library manual coding was rejected. Created software tool generated VHDL (Skahill, 1996) code of any sorting net. Parameters for the tool was: number of inputs, inputs width, serial/parallel, registered/combinatorial. VHDL code was then compiled and implemented by commercial tool. Virtex family of Xilinx's FPGA was targeted technology.

The size of each net was bigger if there was more inputs and wider bit width. There is an exchange of these two parameters if targeted number of digital resources is assumed. If 8 bit values are to be sorted the more inputs are possible when 16 bit values are sorted. If FPGA is target technology optimal utilization of hardware resources is possible because sorting net inputs width exactly fits sorted data width and then number of inputs depend on available number of logic gates. Number of necessary resources to implement different parameters combinatorial, parallel sorting nets is shown in table 2. Resources was expressed in *slices* which are primary reconfigurable blocks of Xilinx Vitrex family (Xilinx, 2003).

Table 2. Number of necessary Virtex resources for combinatorial parallel sorting net

Width →	1 bit	2 bits	4 bits	8 bits
↓ Inputs	Number of slices			
4	6	12	36	84
8	24	48	144	336
16	80	160	480	1120
32	240	480	1440	3360
64	672	1344	4032	9408

For instance for average size Virtex chip XCV1000 number of avaliable slices is 12 288.

Table 3 shows the results when serialization is introduced to the sorting net.

Table 3. Number of necessary Virtex resources for sequential serial sorting net.

Width →	1 bit	2 bits	4 bits	8 bits
↓ Inputs	Number of slices			
4	18	60	90	120
8	72	240	360	480
16	240	800	1200	1600
32	720	2400	3600	4800
64	2016	6720	10080	13440

Crucial parameter of any hardware implementation is delay from input to output time. Table 3 shows delay time for combinatorial realization of various sizes combinatorial sorting nets. Delay time does not depend on inputs bit width because when additional bits are added path from input to output remains the same.

Table 4 Delay time for combinatorial sorting nets. Input width 4 bits. XCV1000 chip

Number of inputs	4	8	16	32	64
Delay time [ns]	30	52	82	119	164

The implementation including registers at every level of ordering elements in sorting net also was performed. As it was mentioned before such implementation allows to achieve better computational performance due to pipelining. It was also stated that such registers increase realisation cost of circuit but it is not the case when FPGA are implementation platform. When register is added at the output of ordering element the number of used slices remains the same. The reason of that is FPGA slice structure. Each slice consist of LUT tables which acts as combinatorial logic element and accompanying each LUT output flip-flop. When combinatorial sorting net is realized flip-flops are bypassed and not used but slice is used itself. Bypassed flip-flops can't by used anymore in the circuit. If registers are not added to the sorting net flip-flops are not used but this not increase the size of available net. It can be treated as if adding registers when FPGA is implementation platform generate no costs. Table 4 shows the implementation cost of parallel sorting net with pipelining implemented by additional registers. For pipelined sorting net data flow is more or less equal despite the sorting net parameters. For Xilinx Virtex family of chips it was 125MHz (maximum path delay between registers is 8ns)

Table 5. Number of necessary Virtex resources for parallel sorting net with pipelining.

Width →	1 bit	2 bits	4 bits	8 bits
↓ Inputs	Number of slices			
4	6	12	36	84
8	24	48	144	336
16	80	160	480	1120
32	240	480	1440	3360
64	672	1344	4032	9408

REFERENCES

Ashenden P. J., The Designer's Guide to VHDL, Morgan Kaufman, 2002

Cormen T. H., Leiserson C. E., Rivest R. L., Introduction to Algorithms, The MIT Press, 2001

Flynn M. J. Very High Speed Computing Systems, Proc. IEEE vol. 54, No. 12, 1966

Hwang K., Advance Computer Architectures. Parallelism, Scalability, Programmability., McGraw Hill, 1993

Knuth D. E., The Art of Computer Programming. Vol 1. Fundamental Algorithms. Addison-Wesley 1998

Omondi, A. R., Computer Arithmetic Systems, Prentice Hall, 1994

Pirsch P., Architectures for Digital Signal Processing, John Wiley & Sons, 1998

Skahill K., VHDL for programmable logic, Prentice Hall, 1996

Wiatr K., Dedicated Hardware Processors for Real-Time Image Data Pre-Processing Implemented in FPGA Structure, Proc. of 9th Int. Conf. on Image Analysis and Processing, Florence 1997, Berlin, Springer-Verlag , 1997, vol II; 69-75

Xilinx: The Programmable Logic Data Book, San Jose CA, Xilinx Inc. 2003