# A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA

Shengnan Dong, Xiaotao Wang,
College of Astronautics
Nanjing University of Aeronautics and Astronautics
Nanjing, China

Xingbo Wang
School of Control Science and Engineering
Shandong University
Jinan, China

***Abstract*—Efficient data sorting is important for searching and optimization algorithms in high time demanding fields such as image and multi-media data processing. To accelerate the data sorting algorithm applied in practical normalized cross-correlation image matching, a novel high-speed parallel sorting scheme based on field programmable gate array (FPGA) is proposed in this paper. When the FPGA chip used has low available logic resource for sorting, the scheme is further extended with random access memory (RAM) as indicators for sorted data to sort large data set. Function and timing simulation with Quartus II 8.0 and practical experiment of the parallel sorting scheme based on the FPGA chip applied in normalized cross-correlation image matching sub-system have shown that this scheme can effectively improve the speed performance with more logic resources usage.**

***Keywords-data sorting algorithm; FPGA; parallel sorting; normalized cross-correlation image matching***

## I. INTRODUCTION

In computer science and mathematics, one of the fundamental problems is ordering a list of items. There's a lot of solutions to this problem, known as sorting algorithms, such as buddle sort, insertion sort, shell sort, merge sort, quick sort, and so forth [1, 2]. Efficient data sorting is important for searching and optimization algorithms in high time demanding fields such as image and multi-media data processing.

Thanks to the rapid development of integrated electronics, the high capability and performance that FPGAs have achieved in last years allow us to accelerate DSP tasks. FPGA devices have been used to implement custom DSPs since the beginning of this decade [3, 4] and have began to be more and more widely used to high time demanding fields as mentioned above.

To accelerate the data sorting algorithm applied in practical normalized cross-correlation image matching, a novel parallel sorting scheme based on FPGA is proposed in this paper to improve the speed performance of the system. At the same time, this scheme can also be used to search for the first $N$ maximum values and sort them in some applications, where $N$ is the required number of the result. When the FPGA chip used has low available logic resource for sorting, the scheme is further extended with RAM as indicators for sorted data to sort large data set. Function and timing simulation with Quartus II 8.0 [5] and practical experiment of the parallel sorting scheme using EP2S60 (the FPGA chip) [6] applied in normalized cross-correlation image matching sub-system have shown that this scheme can effectively improve the speed performance with more logic resources used.

The related principles of the proposed approaches will be introduced in Section II. The simulation results and practical experiment results will be presented in Section III, followed by a conclusion in Section IV.

## II. BASIC PRINCIPLES

### A. Data Sorting Scheme for Short Data Set

The schematic of the parallel data sorting scheme for short data set is shown in Fig. 1, and the corresponding address buffering scheme is shown in Fig. 2. In Fig. 1, the whole processing unit, which can sort $N$ data with $N$ steps and perform data transfer from DFFs (D-flip-flops) to the resultant RAM with the following $N$ steps, consists of $N$ cascade sub-modules including a comparator denoted by *Comparator(n)*, an AND gate, a group of DFFs denoted by *DFF(n)*, and a multiplexer denoted by *Mux(n)*, where $n$ denotes the $n^{th}$ sub-module, $n=1,...,N$. The data inputs to be sorted with required bus width are denoted by *data*. The clock enable signal for data input is denoted by *d_ena*. The outputs of the comparator and the AND gate in the $n^{th}$ sub-module are denoted by *ageb(n)* and *Ena(n)*, respectively. The input and output of the *DFF(n)* are denoted by *d(n)* and *b(n)*, respectively. The whole sorting process is given as follows.

At the beginning, all the outputs of *DFF(n)*, $n=1,...,N$, are initialized to be the minimum representable value. For example, for signed fixed-point data with 8-bits width, the minimum representable value is equal to -127; for unsigned fixed-point data with the same width, the minimum representable value is equal to 0.

At the first step, because the data input to the processing unit are greater than or equal to *b(1)* with initial minimum representable value, the output of the *Comparator(1)* is 1, i.e., *ageb(1)=1*. Accordingly, the output of the *Mux(1)* is equal to *b(1)*, *d(2)=b(1)*. Based on the same principle, *d(2)= b(2)*, *ageb(2) = 1*, therefore *d(3)=b(2)*. Consequently, *d(N) = b(N-1)*, *d(N)= b(N)*, *ageb(N)* = 1. As a result, when *d_ena* is enabled, on the rising edge of the clock input, *data => b(1)*, *b(1) => b(2)*, ... , *b(N-1) => b(N)*, where *x=>y* denotes that the

978-1-4244-4131-0/09/$25.00 ©2009 IEEE

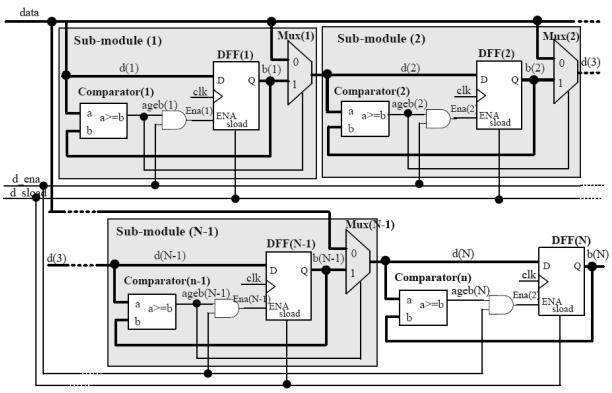


Figure 1. The schematic of the parallel data sorting scheme based on FPGA

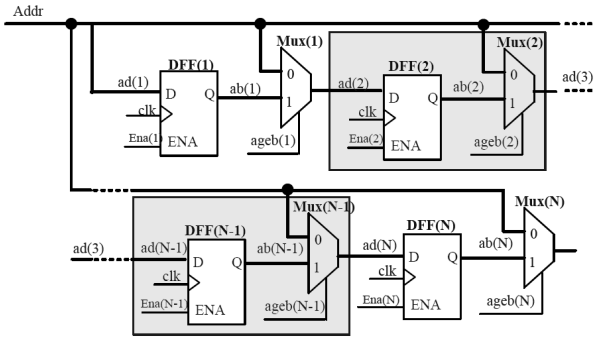


Figure 2. The schematic of the corresponding address buffering scheme based on FPGA

value of *x* will be transferred to *y* on the rising edge of the clock input.

At the second step, if *data >= b(1)*, *ageb(1) = 1*, and similar to the first step, the output of the *Mux(1)* is equal to *b(1),* i.e., *d(2) = b(1)*. As *b(2)* is still the initial minimum representable value, *d(2) > b(2)*, *ageb(2) = 1*. At this time, the status of the following *N-2* sub-modules are similar to the first step. When *d_ena* is enabled, on the rising edge of the clock input, *data=>b(1), b(1) => b(2), ... , b(N-1) => b(N)*. On the contrary, if *data < b(1), ageb(1) = 0*, accordingly, the output of the *Mux(1)* is equal to *data*, i.e., *d(2) = data*. As *b(2)* is still equal to the initial minimum representable value, *d(2) >= b(2)*, *ageb(2) = 1*. The status of the following *N-2* sub-modules are similar to the first step. When *d_ena* is enabled, on the rising edge of the clock input, *b(1)* holds its previous value, d*(2) => b(2), ... , b(N-1) => b(N).*

Consequently, based on the same principle as mentioned above, at the $N^{th}$ step, the *N* data sorted are stored in DFFs. Subsequently, the data sorted in DFFs are shifted to the resultant RAM by the maximum representable value as data input to the processing unit during the following *N* steps. For signed fixed-point data with 8-bits width, the maximum representable value is 127; for unsigned fixed-point data with the same bus width, the maximum representable value is 255.

In conclusion, the whole sorting process including data transfer needs *2N* steps.

### *B. Data Sorting Scheme for Long Data Set*

In the practical normalized cross-correlation image matching sub-system, the first *M*=200 maximum data sorted are required. But logic resource given by the task is low, only *N*=100 cascade sub-modules can be implemented. To solve this problem, a RAM is used to indicate whether the data of the address has been sorted or not. The implementation schematic diagram is shown in Fig. 3.

In Fig. 3, the *Data Sorting Unit* consists of *N* cascade sub-modules as shown in Fig. 1, which can find and sort the first *N* maximum values at one data flow (*M* steps), here, *N<M*. The basic sorting principle given as follows is similar to sorting the short data set.

The processing of the first *N* steps is the same as that of sorting the short data set. Assuming that at the $m^{th}$ step of the processing, all the outputs of the DFFs have been sorted to be *b(1) >= b(2) >= ... >= b(n) >= ... >= b(N-1) >= b(N)*, here, *M>m>N*. The data input (*data*) are less than *b(1), b(2), ..., b(n-1),* and *data >= b(n)*. At this moment, *data < b(1)*, *ageb(1)=0*, the output of the *Mux(1)* is equal to data, i.e., *d(2) = data*, then *d(2)* is also less than *b(2)*, and so on. *d(n-1)=data*, *d(n-1)* is also less than *b(n-1)*, the output of the *Mux(n-1)* is also equal to the *data, d(n) = data*. With the assumption mentioned above, *d(n)>=b(n)*, the output of the *Mux(n)* is equal to *b(n), d(n+1)=b(n)*, and as all the output of the DFFs have been sorted to be *b(1) >= b(2) >= ... >= b(N-1) >= b(N)*, therefore, *d(n+1) >= b(n+1) >= ... >= b(N)*. As a result, *d(n+2) = b(n+1), ..., d(N) = b(N-1)*. When *d_ena* is enabled, on the rising edge of the clock input, *b(1), b(2), ...,* and *b(n-1)* hold their value, *d(n) => b(n)*, *d(n+1)(=b(n)) => b(n+1), b(n+1) => b(n+2), ...,* and *b(N-1) => b(N)*. At the $M^{th}$ step, the first *N* maximum data sorted in DFFs are obtained.

The whole processing principle for sorting the *M* data as shown in Fig. 3 is given as follows. Here, RAM C is a 1-bit RAM used to indicate whether the data of the address has been sorted or not. At the beginning, data are input from the RAM A to the *Data Sorting Unit*. The first *N* maximum values sorted are obtained. At the same time, all the data of the 1-bit RAM C are initialized to be 0. Then the first *N* maximum values sorted and their corresponding addresses in DFFs are shifted to the resultant RAM B by the maximum representable value as data input to the processing unit during the following *N* steps. At the same time, the data in the addresses (corresponding to the first *N* maximum values) of the 1-bit RAM C are set to 1. At the second round, When the indicators in the RAM C of the corresponding addresses to the first *N* maximum values sorted are equal to 1, the first *N* maximum values sorted will be

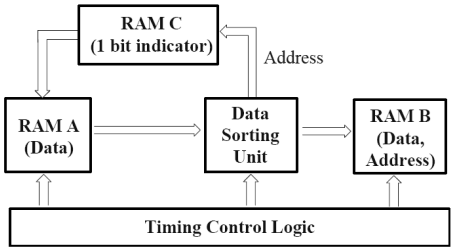


Figure 3. The scheme of data sorting for long data set

disabled, i.e., *d_ena*=0, all the DFFs will also be disabled and hold their values. On the contrary, when these indicators are equal to 0, the *Data Sorting Unit* is enabled, the second *N* maximum values are obtained in the same way as the first round. Consequently, for data set with *M* data, the data sorting is finished in int(*(M+N-1)/N*) steps, int(*z*) is an operator that intercept *z* to integer. The total computing time is (int(*(M+N-1)/N)*M + M*)/*fclk*, where *fclk* is the clock frequency used in the system.

### *C. Computational Complexity and Time Consumed*

The traditional sorting method selects only one maximum value at one data searching process and the whole sorting process needs *(N-2)*(N-1)/2* steps without consideration of data transfer. Compared with the traditional sorting method, the novel scheme has achieved higher speed performance with full use of the high speed and capability advantage of the FPGA for parallel computing. For example, if we have a data set of *M*=100 to be sorted, the traditional sorting method without consideration of data transfer will need (100-2)*(100-1)/2 =4851 steps to finish the whole process. The novel data sorting unit with *N*=100 cascade sub-modules needs 2*100=200 steps. if a novel data sorting unit with only *N*=10 cascade sub-modules is in use, int( (100+9)/10 ) *100 + 100 = 1100 steps for sorting are needed.

## III. SIMULATION AND PRACTICAL RESULTS

### *A. Simulation Results*

In this research, data sorting algorithm was one part of the normalized cross-correlation image matching sub-system. The EP2S60 (the FPGA chip from Altera corporation) [6] was used as the basic development platform. Quartus II 8.0 with SP1 [5] was used to perform logic analysis, synthesis, placement and routing. In data sorting unit, unsigned fixed-point data with 8-bits data width were used as input. The data enable signal was enabled at all the times so that data could be sorted at the speed of the system clock frequency. In this simulation, the first 8 sorted maximum values were obtained at one data flow to demonstrate the effectiveness of the parallel sorting scheme. The compilation report is shown in Fig. 4. From the timing analyzer summary, the system can run at 225.53 MHz.

The timing simulation waveform is shown in Fig. 5. Here, *clk* denotes the system clock signal; *rst_n* denotes the system reset signal; *d_sload* is the signal to enable the load of the initial minimum representable value to the DFFs. *d_ena* denotes data input enable signal. *data* denotes the data to be sorted. *addr* denotes the corresponding address. Eight maximum outputs are denoted by *Max1D, Max2D, ..., Max8D*, where *Max1D >= Max2D >= ... >= Max8D*. The corresponding addresses are denoted by *Max1A_Addr, Max2A_Addr, ... , Max8A_Addr*.

From Fig. 5, the first 8 sorted maximum data outputs and the corresponding addresses are correct. Therefore it is shown that this novel scheme works well.

| | |
|---|---|
| Flow Status | Successful - Thu May 14 10:04:18 2009 |
| Quartus II Version | 8.0 Build 231 07/10/2008 SP 1 SJ Full Version |
| Revision Name | Max4 |
| Top-level Entity Name | Max4 |
| Family | Stratix II |
| Device | EP2S60F672I4 |
| Timing Models | Final |
| Met timing requirements | Yes |
| Logic utilization | < 1 % |
| Combinational ALUTs | 76 / 48,352 ( < 1 % ) |
| Dedicated logic registers | 145 / 48,352 ( < 1 % ) |
| Total registers | 145 |
| Total pins | 180 / 493 ( 37 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 2,544,192 ( 0 % ) |
| DSP block 9-bit elements | 0 / 288 ( 0 % ) |
| Total PLLs | 0 / 6 ( 0 % ) |
| Total DLLs | 0 / 2 ( 0 % ) |

Figure 4. The compilation report of the data sorting unit

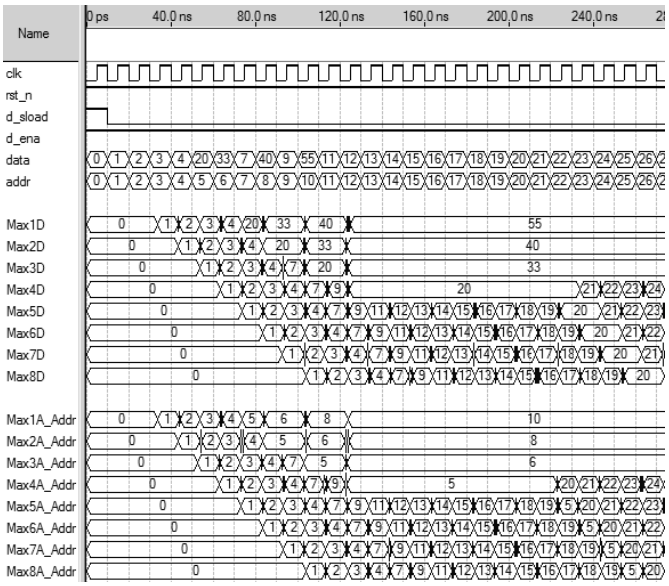


Figure 5. The simulation waveform of the data sorting unit based on FPGA

### *B. Practical Experiment Results*

As one part of the normalized cross-correlation image matching subsystem, the logic implementation of the novel parallel sorting scheme has been applied to the real project. Data are input to the system with a template image of 80*80 and a real-time image of 512*512 with 8-bits data width. So, 433*433(=187489) correlation results can be obtained. The first 200 maximum sorted values were required for further processing.

The block schematic of normalized cross-correlation image matching subsystem is shown in Fig. 6. At the beginning, the digital signal processor - TS201[7] was used to sending the data to the dual-port RAM O and sending image parameters

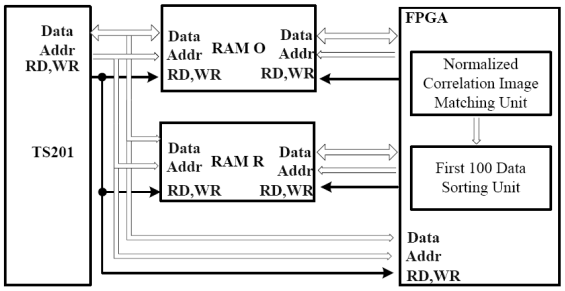


Figure 6. The block schematic of nomalized correlation image matching sub-system

and the start command to FPGA. And then the normalized cross-correlation image matching and the result sorting were started. In FPGA, available logic resource for sorting was low, only 100 cascaded sub-modules was implemented. So during the image matching process, only the first 100 maximum values sorted were obtained. Besides the time consumed by image matching, extra 187489+200=187689 steps are required to search and sort the second 100 maximum value. And the corresponding time consumed is 2.68ms at the system clock of 70 MHz. At the end of sub-system processing, TS201 will receive an interrupt signal to indicate the completeness of the image matching and sorting. To improve system reliability, TS201 will query the corresponding status register of the FPGA to confirm this operation before further processing.

In the practical experiment, the normalized cross-correlation image matching sub-system had achieved desirable image matching performance, and the first 200 maximum values can be sorted correctly. It is shown that the data sorting scheme is feasible and can effectively improve the speed performance.

## IV. CONCULSION

To improve the speed performance of the normalized cross-correlation image matching subsystem, a novel high-speed parallel data sorting scheme based on FPGA is proposed in this paper. When the FPGA chip we used above has low available logic resource for sorting, the scheme is further extended with an additional RAM as indicators for sorted data to sort larger data set. Function and timing simulation and practical experiment results in the real normalized cross-correlation image matching sub-system have shown that this scheme can effectively improve the speed performance with more logic resources used.

## REFERENCES

[1] Sara Baase and Allen Van Golder, Computer Algorithms: Introduction to Design and Analysis, Third Edition, Pearson Education, 2000, pp. 150-221.

[2] Knuth Donald. The Art of Computer Programming, vol. III, Addison-Wesley. 1998, pp. 1–168.

[3] P.J. Graumann, L.E. Turner, “Implementing digital signal processing algorithms using pipelined bit-serial arithmetic and field programmable gate arrays,” First International ACM/SIGDA Workshop on Field Programmable Gate Arrays (FPGA’92), 1992.

[4] M. Wahab and D. Puckey, “FPGA-based DSP Systems,” Eds. W.R. Moore and W. Luk, Abindon EE&CS books, 1994.

[5] Altera Corporation, Quartus II Version 8.0 Handbook, 2008

[6] Altera Corporation, Stratix II Device Handbook, May 2007

[7] Analog Devices. Inc., “TigerSHARC Embedded Processor ADSP-TS201S,” 2006.