http://dx.doi.org/10.5755/j01.eie.22.4.15920 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. 22, NO. 4, 2016

Fast Data Sort based on Searching Networks with Ring Pipeline

Valery Sklyarov[1], Iouliia Skliarova[1], Alexander Sudnitson[2]
[1]*Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, Aveiro, Portugal*
[2]*Department of Computer Engineering, Tallinn University of Technology, Tallinn, Estonia*
skl@ua.pt

***Abstract*—The paper suggests a technique for fast data sort based on a specially organized binary searching network with the following new distinctive features: 1) data sort is done within the time of data acquisition through a serial interface; 2) a new type of pipeline, which we call ring pipeline, is created 3) the delay for receiving each data item is minimized thanks to the novel ring pipeline; 4) sorted data can be transmitted almost immediately after receiving the last input item; 5) several data sets may be sorted by the same network at the same acquisition time. It is proved theoretically that the network is very fast. It was modelled and evaluated in software and the basic components were synthesized and implemented in hardware. The results have shown a significant speed-up comparing to the best known alternatives.**

***Index Terms*—Binary search networks; ring pipeline; block-based merge; communication-time circuits; system-on-chip.**

I. INTRODUCTION

Data sorting is frequently used in different types of information processing. For many practical applications the performance (sorting throughput) is a very important factor [1]. One widely used way enabling the performance to be improved is a hardware/software co-design in which hardware accelerators sort blocks of data and these blocks are further merged in software communicating with hardware through high-performance interfaces. The size N of blocks is limited by the available hardware resources and it rarely exceeds a thousand of words of 16 bits–32 bits. For example, in [2] an 8-element even-odd merging network is built (*i.e.* $N = 8$). In [3] N is increased up to 512 thanks to iterative circuits allowing highly parallel segments with a small propagation delay to be reused. Many other recent publications reviewed in [3] are dedicated to this research area and the majority of the networks are based on either even-odd merge or bitonic merge techniques [4], [5] that have been analysed in detail in [6]. The main idea of the applied methods is processing of a large size set composed of N K-bit vectors and most often $K = 32$ bits. Computations are organized in such a way that $N \times K$ bits (that form each subset) are handled in parallel and, for example, for $N = 256$ and $K = 32$, the network has 8,192 (unsorted) inputs and 8,192 (sorted) outputs. Input/output data cannot be received/transmitted completely in parallel due to limitations either in available pins or in on-chip wires for the very large number (such as 8,192 + 8,192) of external signals. Most often input data are received sequentially through an input port with a typical width equal to 32/64 bits. The result is transmitted through an output port with a similar width. Multiport devices, such as all programmable systems-on-chip (APSoC) from Xilinx Zynq-7000 family permit greater number of signals to be exchanged between software and hardware in parallel, but in any case the maximum number of such signals cannot exceed a few hundreds. Thus, hardware/software sorting involves three sequential stages that are:

1. Receiving $N \times K$-bit unsorted data;
2. Highly parallel processing in a network (such as even-odd or bitonic mergers);
3. Transmitting the results, *i.e.* $N \times K$-bit sorted data.

Note that the indicated above stages may be pipelined, but in any case the points 1) and 3) need to be executed autonomously at least once to get the first unsorted subset and to transmit the last sorted subset. Besides, the effective throughputs of the points 1), 2), 3) are different and an adjustment of stages in the pipeline has to be done, which involves additional hardware resources, restricts capability of transmissions in burst mode and so on finally leading to performance degradation.

The circuit that permits input data to be acquired and sorting these data to be done at the same time is proposed in [7] (see Fig. 1). It contains N K-bit registers $R_0, \ldots, R_{N-1}$, and $N - 1$ comparators/swappers ($\uparrow$) marked with letters a, b, c, d, e, f, g and operating in parallel. For the sake of simplicity, N is assigned to 8. Any comparator/swapper takes two inputs and forms two outputs in such a way that the upper output is greater than or equal to the bottom output, *i.e.* input items are either transferred unchanged or swapped.

At the initialization step, all the registers $R_0, \ldots, R_{N-1}$ are set to the minimum possible value. K-bit data items are received sequentially from an input port through the multiplexer *Mux*. The comparators/swappers move up all input items with non-minimum values, which are accommodated somehow in the registers $R_0, \ldots, R_{N-1}$.

Manuscript received 9 December, 2015; accepted 28 March, 2016.
This research was supported by the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research and Portuguese National Funds through FCT - Foundation for Science and Technology, in the context of the project UID/CEC/00127/2013.

Receiving and accommodating items is done during communication time in N clock cycles. It is shown in [7] that as soon as N data items are acquired and saved in the register, the sorted result can be transmitted immediately from the K-bit output.

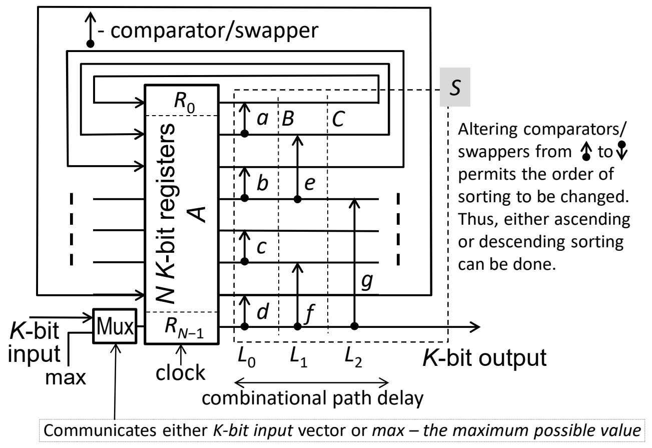
- comparator/swapper
R_0
a
B
C
S
N K-bit registers A
b
e
c
g
d
f
R_{N-1}
K-bit input
Mux
max
clock
L_0
L_1
L_2
K-bit output
combinational path delay
Altering comparators/ swappers from to permits the order of sorting to be changed. Thus, either ascending or descending sorting can be done.
Communicates either K-bit input vector or max – the maximum possible value

Fig. 1. Searching network with feedback for $N = 8$.

Even a superficial comparison of the circuit in Fig. 1 with other known networks permits to conclude that the indicated above stage 2) is completely avoided and the time of sorting is composed of just two stages 1) and 3) that can be pipelined. However, there is a drawback which is similar to the known networks, such as the even-odd and bitonic mergers. The combinational path delay (see Fig. 1) is relatively large, which does not permit to benefit from optimal modes of data transmission. For example, if $N = 256$ then there are $p = 8$ (*i.e.* $p = \lceil \log_2 256 \rceil$) following each other levels of comparators/swappers that involve a delay equal to $8 \times T_{cw}$, where T_{cw} is the delay of one comparator/swapper. As a rule, this delay is not less than a few nanoseconds and thus the combinational path delay is clearly larger than 10 ns. Therefore, data transfer is synchronized by a clock frequency that is less than 100 MHz, which does not permit the full bandwidth capabilities to be supported. The best known even-odd and bitonic mergers involve a delay that is equal to $p \times (p + 1)/2$ [2], *i.e.* for the considered above example the delay is increased from $8 \times T_{cw}$ to $36 \times T_{cw}$.

We suggest below further improvements through the following additional features: 1) introducing ring pipeline that enables data to be received with significantly higher clock frequency; 2) using the same circuit for sorting several blocks at the same time for their subsequent merging in software and permitting burst sizes for the input port to be increased; 3) transmitting the sorted results with higher frequency thanks to the proposed ring pipeline.

II. NETWORK ARCHITECTURE AND FUNCTIONALITY

Let us introduce a ring pipeline, *i.e.* let us accommodate additional registers in places indicated in Fig. 1 by dashed vertical lines B, C, *i.e.* between the comparators/swappers that are active at each level. Now the circuit in Fig. 1 may handle several subsets with N items each. The primary $N \times K$-bit register A, shown in Fig. 1, and the newly introduced registers B, C form a 3-stage ring pipeline. The number of such stages for a general case is equal to $p = \lceil \log_2 N \rceil$ and the relevant circuit enables p sets with N K-bit elements to be processed. Let us consider an example for $N = 4$ and a 2-stage ring pipeline ($p = \lceil \log_2 N \rceil = 2$). Now $p = 2$ subsets F_1 and F_2 can be sorted passing through two registers A and B linked to a ring. Suppose: $F_1 = \{36_1, 101_1, 102_1, 84_1\}$ (subscript 1 indicates belonging to the subset 1); $F_2 = \{116_2, 42_2, 62_2, 41_2\}$ (subscript 2 points to the subset 2). For the considered example 8 clock cycles will be involved enabling the two subsets F_1 and F_2 to be sorted. States of the registers converted by the relevant comparators/swappers in each clock cycle 1), …, 8) are the following: 1) input 36_1: $A = \{0, 0, 36, 0\}$, $B = \{0, 0, 0, 0\}$; 2) input 116_2: $A = \{0, 0, 116, 0\}$, $B = \{0, 0, 36, 0\}$; 3) input 101_1: $A = \{0, 0, 101, 36\}$, $B = \{0, 0, 116, 0\}$; 4) input 42_2: $A = \{0, 0, 116, 42\}$, $B = \{0, 36, 101, 0\}$; 5) input 102_1: $A = \{36, 0, 102, 101\}$, $B = \{0, 42, 116, 0\}$; 6) input 62_2: $A = \{42, 0, 116, 62\}$, $B = \{36, 101, 102, 0\}$; 7) input 84_1: $A = \{101, 36, 102, 84\}$, $B = \{42, 62, 116, 0\}$; 8) input 41_2: $A = \{62, 42, 116, 41\}$, $B = \{101, 84, 102, 36\}$. After the cycle 8) two sorted subsets can be transmitted through a single output port (see Fig. 1) also in 8 clock cycles as follows: 1) 36_1: $A = \{101, 84, \max, 102\}$, $B = \{62, 42, 116, 41\}$; 2) 41_2: $A = \{62, 42, \max, 116\}$, $B = \{\max, 102, 101, 84\}$; 3) 84_1: $A = \{\max, 102, \max, 101\}$, $B = \{\max, 116, 62, 42\}$; 4) 42_2: $A = \{\max, 116, \max, 62\}$, $B = \{\max, 102, \max, 101\}$; 5) 101_1: $A = \{\max, \max, \max, 102\}$, $B = \{\max, 116, \max, 62\}$; 6) 62_2: $A = \{\max, 116, \max, \max\}$, $B = \{\max, \max, \max, 102\}$; 7) 102_1: $A = \{\max, \max, \max, \max\}$, $B = \{\max, \max, \max, 116\}$; 8) 116_2: $A = \{\max, \max, \max, \max\}$, $B = \{\max, \max, \max, \max\}$. Finally, we get two sorted subsets $F^s_1 = \{36_1, 84_1, 101_1, 102_1\}$ and $F^s_2 = \{41_2, 42_2, 62_2, 116_2\}$. As you can see only the transmission time is involved and sorting is done in parallel with transmitting data and does not involve any additional delay.

Note that the circuit in Fig. 1 implements binary search and can easily be cascaded as it is shown in Fig. 2, where S is the segment from Fig. 1. Now there are 5 pipeline registers ($p = \lceil \log_2 N \rceil = 5$) that are the main register A, the registers B and C in the segments S and the registers D and E (see Fig. 2). The circuit in Fig. 2 can sort 5 subsets (with $N = 32$ K-bit elements each one) in $N \times p = 160$ clock cycles and output the results also in $N \times p = 160$ clock cycles. Note that receiving data items through a K-bit channel requires 160 clock cycles and as soon as the last item is received the sorted results can be transmitted immediately. Similarly, networks with a ring pipeline can be built for larger values of N. Since the network implements binary search the number of comparators/swappers in the first level (marked as L_0 in Fig. 1) is equal to $N/2$. The number of comparators/swappers in each subsequent level is changed as follows: $N/2^2$, ..., $N/2^p$. Thus, the total number $C(N)$ of comparators/swappers is equal to: $\sum_{n=1}^{\lceil \log_2 N \rceil} N/2^n$. The number of levels in networks [7] is p. So, the delay is $p \times T_{cw}$. Thus, any new input item can be received with the delay $p \times T_{cw}$ required for transmitting input signals through p combinational levels. In the proposed circuit with a ring pipeline any new input item can be received with the delay T_{cw}, *i.e.* faster by a factor of p. Besides, since many subsets can be sorted by the same circuit, the number β of received items is increased by a factor of *p*. To improve efficiency of getting input data a burst mode is commonly applied. The size of the burst can influence the speed of data transmission and enlarging this size often allows the speed to be increased. Thus, the proposed method that permits larger number of data items to be subsequently transmitted enables port throughput to be augmented.

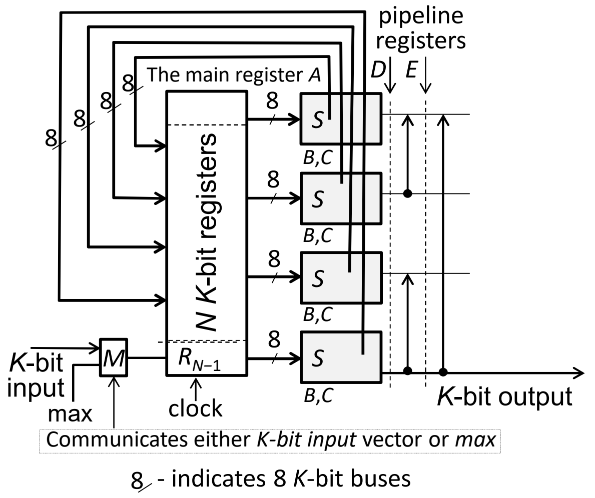

Fig. 2. An example of a cascaded searching network for *N* = 32.

Note that in the proposed method input items are acquired as follows: 1) in the first *p* steps only the first items from *p* given sets are received by the network (let us call these steps group 1); 2) in the second *p* steps the second items from *p* given sets are acquired by the network (let us call these steps group 2); *etc*. until the final group *p*. Such type of data transmission can easily be organized. For example, we made experiments with Zynq-7000 microchips available on the prototyping board [8]. Software, running in the processing system of the Zynq-7000 device [8], copies data to a shared memory in such a way that at the beginning of each memory segment the first items from *p* given sets are saved, then the second items from *p* given sets are saved and so on. Hence, the programmable logic takes data from the shared memory sequentially and transmits the sorted subsets also sequentially. It is easily visible from the example above that the sorted data are organized similarly, *i.e.* at the beginning the first items from all *p* sorted subsets are transmitted, then the second items from all *p* sorted subsets are transmitted, *etc*. Hence, the results of sorting are presented in the same form as input (unsorted) data. Subsequent merging (that is frequently involved for processing large data sets [2], [3]) is done in software much like explained in [3].

TABLE I. THE REQUIRED NUMBER $C(N)$ OF COMPARATORS/SWAPPERS FOR DIFFERENT SORTING NETWORKS.

Network type	$C(N)$, $p = \lceil \log_2 N \rceil$
Bubble and insertion sort	$N \times (N-1)/2$
Even-odd transition	$N \times (N-1)/2$
Even-odd merge	$(p^2 - p + 4) \times 2^{p-2} - 1$
Bitonic merge	$(p^2 + p) \times 2^{p-2}$
The proposed in this paper	$\sum_{n=1}^{p} N / 2^n$

Let us compare theoretically the complexity of the proposed and the most frequently used known networks. A traditional (*i.e.* not of the proposed *ring* type) pipeline can be used for the known sorting networks. Thus, the delay for the known networks may become exactly the same as the delay for the proposed network with the ring pipeline (see Fig. 2). Hence, the preference of a particular network can be done by comparing the required resources. Table I shows such resources for different networks including the proposed one. The relevant formulae are taken from [2], [3], [9].

Charts in Fig. 3 permit the numbers $C(N)$ to be compared for various networks and different values of *N*.

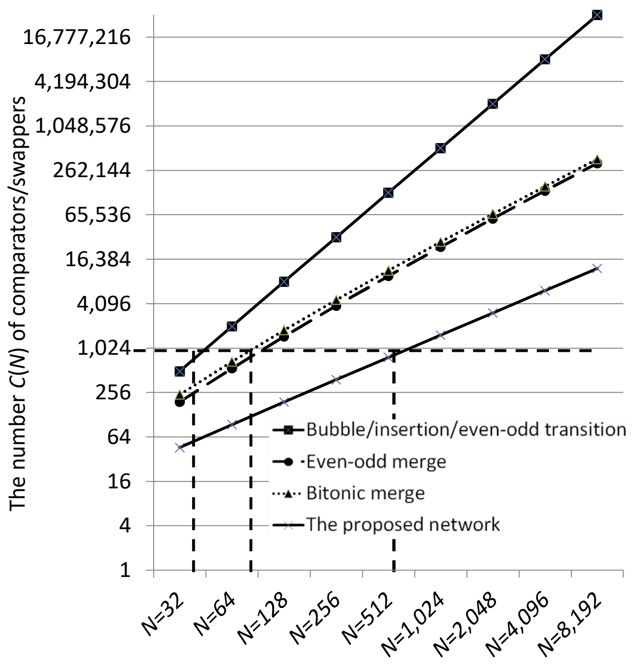

Fig. 3. Theoretical evaluation of the required resources for the proposed and known networks.

It should also be noted that the number of pipeline registers in the proposed network is always less than for any known network because the number of propagation stages in the proposed networks is smaller (compare, for instance, the number of stages 8 and 36 for *N* = 256 in the example of Section I). Besides, neither from the known networks is able to sort data during communication time. Thus, the proposed networks are faster and less resource consuming.

III. SIMULATION IN SOFTWARE AND IMPLEMENTATION IN HARDWARE

The functionality of the network *S* in Fig. 1 can be tested by the following function SN written in the Java language:

```
public static int SN(int a[])     {     // N and p are
constants
   int tmp;
   for(int k = 0; k < p; k++)
      for(int i = 0; i < a.length/s[k+1];  i++)
         if (a[s[k+1]*i+s[k]-1] < a[s[k+1]*i+s[k+1]-1])
         {        tmp = a[s[k+1]*i+s[k]-1];
                  a[s[k+1]*i+s[k]-1] = a[s[k+1]*i+s[k+1]-1];
                  a[s[k+1]*i+s[k+1]-1] = tmp;              }
      return a[N-1];          }
```

where the array a can be generated randomly in the function main, for example:

```
    int[] a = new int[N];  // N is the number of elements
    for(int x = 0; x < N; x++)
        a[x] = rand.nextInt(Integer.MAX_VALUE);
```

The array s is declared as a static member in the main class as **static final int** s[] = {1,2,4,8,/*…2^p */};, for example, for $N = 16$: **static final int** s[] = {1,2,4,8,16};. The object rand is declared as follows: **static** Random rand = **new** Random();.

The complete circuit in Fig. 1 is modelled in Java through the sequential reading *K*-bit inputs from the port, which is simulated by a sequential generation of an arbitrary *K*-bit value. It is done as follows:

```
    for(int x = 0; x < N; x++)
        {   a[N-1] = rand.nextInt(Integer.MAX_VALUE);
// ……..
```

where a[N-1] is the value coming from the Mux in Fig. 1.

The Java code was converted to a VHDL specification. For example, the function SN above is presented as follows:

```
process(data_in) -- data_in is an input port declared as
-- follows: data_in : in  std_logic_vector(N*K-1 downto
0);
            variable MyAr : in_data;
            variable tmp :  std_logic_vector(K-1 downto
0);
            begin  -- K is the number of bits for each
element
                for i in N-1 downto 0 loop
                    MyAr(i) := data_in(K*(i+1)-1 downto
K*i);
                end loop;
                for k in 0 to p-1 loop
                    for i in 0 to N/(2**(k+1))-1 loop
                        if ( MyAr( 2**(k+1)*i+(2**k)-1 ) <
                        MyAr(  2**(k+1)*i+2**(k+1)-1  ) )
                    then
                            tmp          :=          MyAr(
2**(k+1)*i+(2**k)-1 );
                            MyAr( 2**(k+1)*i+(2**k)-1 ) :=
                              MyAr(  2**(k+1)*i+2**(k+1)-1
                            );
                            MyAr(  (2**(k+1)*i+2**(k+1)-1)
) := tmp;
                        end if;
                    end loop;
                end loop;
                min_value  <= MyAr(N-1);
            end process;
```

The type in_data is declared as follows:

```
    type in_data is array (N-1 downto 0) of
        std_logic_vector(K-1 downto 0);
```

Supplying input data from one port is done in VHDL much like as it is shown in the Java fragment above.

The ring pipeline (see Fig. 2) is modelled in the Java program using a two-dimensional array a[i][j]. The first dimension permits to work with *p* registers of the ring pipeline a[0], …, a[p-1]. The second dimension enables to access bits of each *N*-item register indicated by the index of the first dimension. For example, two sorted subsets shown above ($F^s_1 = \{36_1, 84_1, 101_1, 102_1\}$ and $F^s_2 = \{41_2, 42_2, 62_2, 116_2\}$) will be modelled by the array a with two elements for the first dimension (for the subsets F^s_1 and F^s_2) and with four elements for the second dimension (for 4 elements of each subset). The program permits the results of networks with different numbers of *N* to be verified. For instance, the results of the example from Section II may be displayed or saved in a file as follows:

```
36                              116
   0;    0;   36;    0;            0;    0;  116;    0;
   0;    0;    0;    0;            0;    0;   36;    0;
101                             42
   0;    0;  101;   36;            0;    0;  116;   42;
   0;    0;  116;    0;            0;   36;  101;    0;
102                             62
  36;    0;  102;  101;           42;    0;  116;   62;
   0;   42;  116;    0;           36;  101;  102;    0;
84                              41
 101;   36;  102;   84;           62;   42;  116;   41;
  42;   62;  116;    0;          101;   84;  102;   36;
-------------------------------------------------------
  36;   84;  101;  102;
-------------------------------------------------------
  41;   42;   62;  116;
-------------------------------------------------------
```

The sorted items for two subsets are shown in between three horizontal dashed lines. Similar results of any example with *N* items can be verified formally through comparison of all the values in the results.

Finally, the proposed network with a ring pipeline (see Fig. 2) was evaluated in software and tested in hardware [8]. The software model proves that the network in Fig. 2 functions as expected. The hardware model verifies the correctness of theoretical evaluations and permits the proposed networks to be compared with the known networks. The results of experiments demonstrate that the proposed networks are faster and less resource consuming with almost the same factors that were given above (see Section II) for theoretical evaluation. We checked also an opportunity for using multiple ports, such as that are available in Zynq-7000 devices, and found that the proposed network (see Fig. 2) is well suited for grouping input/output data that enables several items to be transmitted in parallel.

IV. CONCLUSIONS

The paper suggests a novel hardware solution for data sorting that is based on a ring pipeline. The main advantages of the proposed technique are: 1) sorting is done at the time of acquiring input data items through either a single or multiple ports; 2) the speed of input interface is raised thanks to the proposed ring pipeline with reduced number of stages comparing to the known networks; 3) the sorting network does not involve any additional delay. These results have been confirmed both theoretically and experimentally.

REFERENCES

[1] A. Rjabov, V. Sklyarov, I. Skliarova, A. Sudnitson, "Processing sorted subsets in a multi-level reconfigurable computing system", *Elektronika ir Elektrotechnika*, vol. 21, no. 2, pp. 30–33, 2015. [Online]. Available: http://dx.doi.org/10.5755/j01.eee.21.2.11509

[2] R. Mueller, J. Teubner, G. Alonso, "Sorting networks on FPGAs", *The Int. Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 1–23, 2012. [Online]. Available: http://dx.doi.org/10.1007/ s00778-011-0232-z

[3] V. Sklyarov, I. Skliarova, "High-performance implementation of regular and easily scalable sorting networks on an FPGA", *Microprocessors and Microsystems*, vol. 38, no. 5, pp. 470–484, 2014. [Online]. Available: http://dx.doi.org/10.1016/ j.micpro.2014.03.003

[4] K. E. Batcher, "Sorting networks and their applications", in *Proc. AFIPS Spring Joint Computer Conf.*, 1968, pp. 307–314. [Online]. Available: http://dx.doi.org/10.1145/1468075.1468121

[5] S. W. Aj-Haj Baddar, K. E. Batcher, *Designing Sorting Networks. A New Paradigm*. New York: Springer-Verlag, p. 136, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1851-1

[6] D. E. Knuth, *The Art of Computer Programming, Sorting and Searching, vol. III*. Addison-Wesley, p. 896, 1998.

[7] V. Sklyarov, I. Skliarova, "Hardware accelerators for data sort in all programmable systems-on-chip", *Advances in Electrical and Computer Engineering*, vol. 15, no. 4, pp. 9–16, 2015. [Online]. Available: http://dx.doi.org/10.4316/AECE.2015.04002

[8] Avnet, Inc., *ZedBoard (Zynq^{TM} Evaluation and Development) Hardware User's Guide*, Version 2.2, 2014. [Online]. Available: http://www.zedboard.org/sites/default/files/

[9] P. Kipfer, R. Westermann, "Improved GPU sorting", *GPU Gems 2*, p. 880, 2005. [Online]. Available: http:// http.developer.nvidia.com/GPUGems2/gpugems2chapter46.html