http://dx.doi.org/10.5755/j01.eie.23.3.18336 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. 23, NO. 3, 2017

Fast Processing of Non-Repeated Values in Hardware

Iouliia Skliarova[1], Valery Sklyarov[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*
iouliia@ua.pt

Abstract—**The paper suggests a technique for fast data processing of unique and constrained items. The technique is based on two methods that involve: 1) address-based data sorting; and 2) communication-time networks. Input data are received one by one from a sequential channel. The first method enables undesirable values (e.g. previously taken or explicitly blocked) to be discarded. Although this method is chosen from the scope of data sorting, it is used in the paper (after some adjustments) for filtering. The second method enables each data item to be properly handled during communication time. For example, in case of data sorting it means that as soon as a new item is received it will immediately be placed in a proper position of the produced sorted subset that is composed of all previously acquired items. The circuits that implement the proposed methods have been entirely modeled and verified in software, then described in VHDL, synthesized and implemented in hardware, and finally evaluated. The results have shown that the proposed solutions are well suited for real-time applications.**

Index Terms—**Address-based sorting; data filtering; sorting networks; communication-time circuits; FPGA.**

I. INTRODUCTION

Data processing is one of the most common procedures in computational systems. Examples of such processing are data sorting/searching, discovering of frequent items, filtering and so on. Very often different operations over data are implemented in highly parallel networks that take input values and convert them to output values in a combinational circuit. The output values are not formed immediately because input signals pass through logic elements each of which causes a delay on a way of the signals from the inputs to the outputs. In many practical applications we would like to handle just non-repeated values, i.e. each of such values is unique. Let us consider, for instance, a set of natural values: {4, 0, 18, 5, 6, 4, 3, 3, 4} and some of them (e.g. the values 3 and 4) are repeated. Hence, the set of unique or non-repeated values is {4, 0, 18, 5, 6, 3}. For many practical applications that can be found in cryptography, bioinformatics, feature extraction, data mining and some other areas, we would like to separate out only unique values and, besides, even some of such values need to be discarded, i.e. they are not allowed to be taken for the subsequent processing. For instance, in the set with unique values, given above {4, 0, 18, 5, 6, 3}, we may block some interval, for example, the values greater than or equal to 5 and less than or equal to 6. With such the requirement the set above is reduced to the set: {4, 0, 18, 3}. In many cases this task needs to be solved in real time, data processing has to be very fast and, thus, hardware accelerators are necessary. The paper suggests one possible solution of this problem through an integration of two methods involving procedures from address-based data sorting and communication-time or real-time data processing. The first method, which is proposed in [1], considers the value v of each data item as the memory address on which this memory contains a one-bit flag indicating whether the value v is new or has already been received. Let us assume that data items are taken sequentially. It is shown below that the method [1] permits very fast detection of unique (non-repeated) items. To satisfy requirements of real-time applications, processing has to be done as fast as possible. The type of processing itself may vary but most often we execute such procedures as sorting, searching, and filtering (i.e. extracting items with the desired characteristics). The majority of such problems can be solved in two of the most frequently investigated highly-parallel circuits based on sorting [2] and linear [3] networks. The majority of sorting networks implemented in hardware use Batcher even-odd and bitonic mergers [4]. There are limitations for both circuits [4] and [3] because they either introduce a very long propagation delay in [4] or are only suitable for a very small number of items in [3]. This conclusion is also valid for partially combinational and partially sequential networks described in [5]. Such limitations make it difficult to apply the mentioned above methods to real-time data processing. We suggest such processing to be executed on circuits [6] enabling data to be handled during communication time. Thus, two major innovations of this paper are the following:

1. Detecting non-repeated and other constrained values with the aid of a method inherited from the address-based data sorting [1].
2. Fast data processing based on communication-time networks that allow both sequential data acquisition and processing (e.g. sorting) to be done in parallel. For

Manuscript received 30 November, 2016; revised 9 March, 2017.
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.

instance, if the main objective is data sorting then we would like to get all the received data sorted immediately after the last item has been taken from the input.

The indicated above two major innovations are discussed in Sections II and III, accordingly.

II. DATA FILTERING

The objective of this section is to describe the proposed simple, but efficient method that enables an input sequence to be filtered and the output sequence to be formed in such a way that:

1. All the source items are included in the output sequence just once. If there are several items with the same value then just one of them is included.
2. All predefined and explicitly indicated values are removed from the output sequence.

The proposed circuit is created on the basis of [1]. Its functionality is explained below on an example of modeling of the circuit in the following Java program:

```
import java.io.*;
public class Generate_Non_repeated {
        static Random rand = new Random();
        final static int N = 32;
        final static boolean[] for_test = new boolean[N];

public static void main (String args[])
{       int[] nr = new int[N];
        clean();
        for(int i = 5; i < 10; i++) block(i);
        for(int i = 0; i < N; i++)
            nr[i] = rand.nextInt(16);
        for(int i = 0; i < N; i++)
            if (verify_rep(nr[i]))    // transfer to the output
                                      // and process the item
}
public static boolean verify_rep(int check_me)
{       if (for_test[check_me] == true) return false;
        else for_test[check_me] = true;
        return true;                          }
public static void clean()
{       for(int i = 0; i < N; i++) for_test[i] = false;   }
public static void block(int b)
{       for_test[b] = true;   }
```

The program above generates randomly a set of integers that are included in the array «nr». We declared also an additional array «for_test» with Boolean values. This array is used much like the memory for sorting data in [1]. All the array elements are initially set to **false** value by the «clean» function. In fact, it is done automatically in software but we would prefer to invoke the function «clean» just to not forget cleaning in hardware if required. The function «block» allows some undesirable values to be blocked. As an example, we eliminated all the values greater than or equal to 5 and less than 10. The function «verify_rep» tests each incoming value v and if this value appears for the first time then on the address which is the same as the value v the value **true** is written, otherwise the incoming value v has already been processed previously and, thus, the new repeated value is discarded. For the sake of simplicity, the input sequence is generated randomly and some particular limitations are settled. Clearly, the input sequence and the limitations may easily be changed.

Figure 1 demonstrates the proposed filtering circuit, which is very simple and fast.

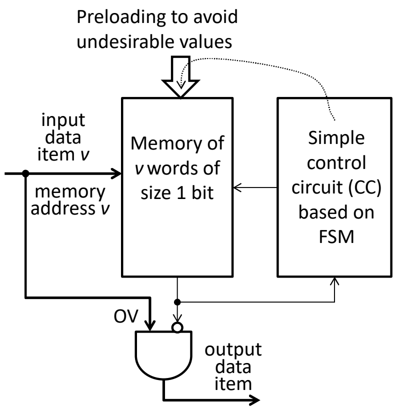

Preloading to avoid undesirable values
input data item v
memory address v
Memory of v words of size 1 bit
Simple control circuit (CC) based on FSM
OV
output data item

Fig. 1. The proposed filtering circuit (OV is the original value).

The core of the circuit is the memory that has v 1-bit words. If the value 1 is written to an address v then the value v is blocked, i.e. it cannot be included in the output sequence, and otherwise the value v has to be included in the output sequence. A simple control circuit (CC) is a finite state machine (FSM), which preloads values 1 to the memory to such addresses for which the corresponding values v have to be discarded and sets the value 1 for the first value v in the input sequence. The bottom AND gate (a trivial multiplexer) either allows (when on the address v the flag 0 is stored) or not (when on the address v the flag 1 is stored) the value v to be transferred to the output.

The circuit in Fig. 1 is very fast. Preloading is executed only during an initialization phase. Data are received and the output is generated with the speed of access to the memory. The details of hardware implementation will be discussed in Section IV.

Note that a trivial modification of the circuit in Fig. 1 enables data items to be avoided if they fall in an interval of close values. Let us assume that any input value of size M bits can be handled ($M = \lceil \log_2 v \rceil$). If we take only the most significant $M-k$ bits as memory addresses in Fig. 1, then the interval is composed of 2^k close values. For example if k = 2 then the intervals are {0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}, etc. The value 1 in the memory written to the address v for our example denotes that one of the values v, $v+1$, $v+2$, $v+3$ has already arrived and, thus, any subsequent value v, $v+1$, $v+2$, $v+3$ will be discarded. Hence, even a very simple circuit shown in Fig. 1 is powerful enough and can be used in many practical applications.

III. DATA SORTING

We demonstrate communication-time data processing on an example of data sorting. At the beginning let us analyse the circuit in Fig. 1. In [1] a similar circuit was used for sorting data items. Indeed, as soon as all input data have been received the values 1 in the memory permit to find the sorted sequence with unique values. This sequence is extracted by sequential reading the memory beginning from the address 0 and forming the sorted data from the addresses for which the values in the memory are 1. However, v clock cycles would be needed for such type of processing. Since we would like to sort the input sequence in real-time, the elaborated in [1] solution cannot be chosen because it is relatively slow. That is why we propose another solution that is shown in Fig. 2.

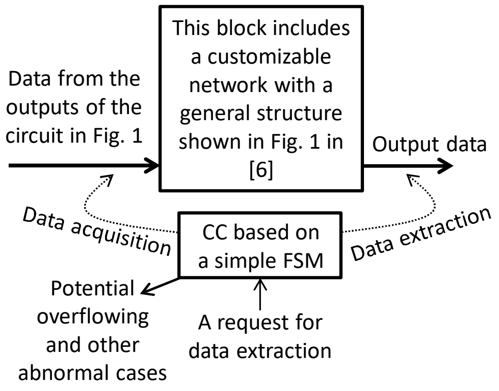

This block includes a customizable network with a general structure shown in Fig. 1 in [6]
Data from the outputs of the circuit in Fig. 1
Output data
Data acquisition
CC based on a simple FSM
Data extraction
Potential overflowing and other abnormal cases
A request for data extraction

Fig. 2. Communication-time data processing.

The largest block in Fig. 2 contains the circuit that permits input data to be acquired and ordered. It includes a register and a network connected to the register outputs. Outputs of the network are connected once again to some inputs of the register. All necessary details can be found in [6]. As soon as a new data item arrives from the input (i.e. from the output of the circuit in Fig. 1), it is placed in a certain word of the register. This way permits at any time to begin copying the sorted data items. Thus, any pre-filtered data item received on the input of the circuit in Fig. 2 can be accommodated within the subset that contains all the previously received data items so that the sorted data from the subset can be sequentially transferred to the output of the circuit in Fig. 2 in either ascending or descending order depending on customization of the network [6]. The Java program below enables complete modeling of the larger block in Fig. 2 to be done.

```
import java.util.*;
public class run_time
{   public static final int p = 3;
    public static final int N = (int)Math.pow(2,p);
    static int[] from_channel = {30,17,41,11,5,4,40,71};
    static final int s[] = {1,2,4,8};

public static void main(String[] args)
{   int a[] = new int[N];
    print_sorted(SN(a));   }

public static int[]  SN(int a[])       { // receiving and
int tmp, j=0;                            // accommodating data
for(int x = 0; x < N; x++){
   a[N-1] = from_channel[j++];
   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;               }
           for(int ii = 0; ii < a.length;  ii++)
         {  System.out.printf("%5d; ",a[ii]);
              if (((ii+1)%N) == 0) System.out.println();    }
   }
return a;             }
```

```
public static void print_sorted(int a[])  { // extracting
int tmp, out;  System.out.println();     // the sorted data
for(int x = 0; x < N; x++)                  {
   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;             }
           out = a[N-1]; a[N-1] = Integer.MAX_VALUE;
         System.out.printf("%5d;",out);     }
      System.out.println();
   }
}
```

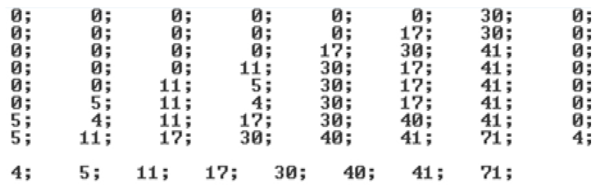
The array **s** is declared to be a static member in the main class as: **static final int** s[] = {1,2,4,8,/*…2^p */};, for example, for p = 3 the declaration becomes the following: **static final int** s[] = {1,2,4,8};. Fig. 3 demonstrates the results of the program execution for the following (arbitrary chosen) input data:

```
static int[] from_channel = {30,17,41,11,5,4,40,71};
```

Clearly, the number of input data (i.e. from_channel.length) and their values (i.e. {30,17,41,11,5,4,40,71}) may easily be changed. Additional example is given below:

```
public static final int p = 4;
static int[] from_channel =
 {30,17,41,11,5,4,40,71,330,7,401,111,55,9,16,39};
static final int s[] = {1,2,4,8,16};
```

where all necessary changes are shown to allow the number of data items to be changed from 8 to 16. The values of data items may be chosen arbitrary.



```
   0;      0;      0;      0;      0;      0;     30;      0;
   0;      0;      0;      0;      0;     17;     30;      0;
   0;      0;      0;      0;     17;     30;     41;      0;
   0;      0;      0;     11;     30;     17;     41;      0;
   0;      0;     11;      5;     30;     17;     41;      0;
   0;      5;     11;      4;     30;     17;     41;      0;
   5;      4;     11;     17;     30;     40;     41;      0;
   5;     11;     17;     30;     40;     41;     71;      4;

   4;     5;    11;    17;    30;    40;    41;    71;
```

Fig. 3. The results of the Java program execution.

Figure 3 shows all the steps of accommodating input data items in the major block of Fig. 2 for the given sequence. The leftmost item (30 in our example) is always the first. The second item (17 in our example) is the second and so on. Lines of Fig. 3 (from the top to the bottom) show accommodation in the register [6] of input data after a new item has arrived. For example, the third line (0 0 0 0 17 30 41 0) shows the contents of the register [6] after three items 30, 17, and 41 have arrived. If we want to output data after this step the output data will be transferred in the sorted order, i.e. 41, 30, 17. Figure 3 presents the result (see the bottom line) after receiving all 8 items from the input set from_channel. If we link the first and the second program, we can model in software the complete functionality of the circuits proposed in this paper. Similar ideas may also be used for other types of data processing.

IV. IMPLEMENTATION IN HARDWARE

The presented above Java programs have been converted to VHDL specifications much like it is done in [6] using also newly available tools described below. Synthesis and implementations were done in the Vivado Design Suite of Xilinx for the Nexys-4 prototyping board of Digilent [7]. We created a user-defined IP (Intellectual Property) core for data sorting (see Section III) in VHDL. The circuit for filtering (see Section II) is built in the Vivado IP block integrator and the used memory is a dual-port RAM from the system repository. This circuit is considered to be the second user-defined IP. The top-level circuit was also designed in the Vivado IP block integrator in such a way that the IP cores for filtering and data sorting are taken from the user repository. The experiments have shown that each incoming data item is handled within the same time that is required to read data from the dual-port RAM. An additional multiplexer (see the bottom part of Fig. 1) adds an insignificant delay. In the experiments, the size M of data items is chosen to be 16 bits (it can easily be changed through generic constraints). Thus, just one built-in FPGA RAM is sufficient to create the memory for the circuit in Fig. 1. Additional logic for this circuit occupies a negligible part of the available FPGA resources. The circuit in Fig. 2 is significantly more resource consuming. However, more than one thousand data items can be handled even in the relatively simple FPGA of the Nexys-4 board. Greater number of data items can either be processed in an advanced FPGA or a decomposition technique (see, for example, sorting plus merging in [2]) can be used.

Note that the described above data processing has been applied to only natural values that are greater than or equal to zero. However, real numbers may also be processed with the proposed technique. The only problem can appear for the circuit in Fig. 1 because in this case it is substantially more difficult to detect non-repeated values. One possible way is to consider for each real value v the nearest integer instead of v. This obviously leads to the loss of precision. However, instead of original values we can deal with shifted values. For example, in case of a decimal real value v we can shift left the value v before the processing and handle again the nearest integer instead of the real value v. Let $v = 12.125$. If we shift this value left by one decimal position then the given value v will be changed to the value 121.25. Finally, we will check the previously received values to which the same type of shifting and conversion has been applied. Clearly, processing the shifted values is more precise than processing values rounded to the nearest integer. Likewise, we can shift the original values by more than one digit and achieve even better precision. Negative numbers can easily be presented in the form of natural numbers. Let x be the maximum absolute value of negative integers. We can add x to the actual values v and handle natural numbers for negative values. Note that the communication-time circuit from [6] can sort real and negative values.

Figure 4 depicts additional circuits that are needed to handle real and negative numbers. The circuit in Fig. 4 executes pre-processing to convert negative numbers to positive numbers and permits real numbers to be used. The pre-processing circuit in Fig. 4 implements the described above transformations of data items enabling the circuit in Fig. 1 to be utilized much like as before. Generally speaking, negative and real numbers are transformed to some integers that are identical for data items that are checked for repetition or discarding. Note that the original values (see OV in Fig. 4) are transferred to the input of the multiplexer (see the bottom part of Fig. 1 marked with OV).

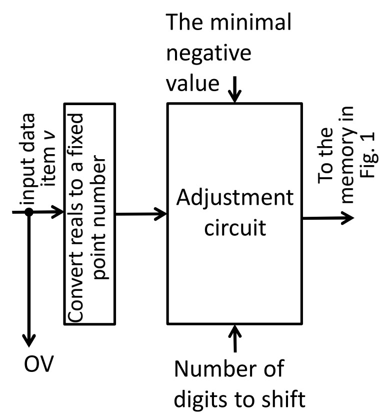

The minimal negative value
input data item v
Convert reals to a fixed point number
Adjustment circuit
To the memory in Fig. 1
OV
Number of digits to shift

Fig. 4. Pre-processing of real and negative numbers.

V. CONCLUSIONS

The paper is dedicated to fast processing of non-repeated values some of which can be discarded on request. The emphasis is done on real-time applications in which data items have to be received and processed with a minimal possible delay. Several tasks have been discussed. The objective of the first task is to avoid repeated values in the input set that needs to be processed. The second task analyses not only the exact value but also a set of the closest values within the defined interval. The last task permits additional predefined values to be also eliminated. The proposed solutions are modelled and carefully evaluated in software programs and implemented in hardware (in FPGA). The experiments have demonstrated that the suggested circuits enable data processing with the speed of data transfer and are therefore applicable for real-time systems.

REFERENCES

[1] V. Sklyarov, I. Skliarova, D. Mihhailov, A. Sudnitson, "Implementation in FPGA of address-based data sorting", in *Proc. 21st Int. Conf. on Field Programmable Logic and Applications*, Crete, Greece, 2011, pp. 405–410. [Online]. Available: http://dx.doi.org/10.1109/FPL.2011.81

[2] R. Mueller, J. Teubner, G. Alonso, "Sorting networks on FPGAs", *The International 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] J. Ortiz, D. Andrews, "A streaming high-throughput linear sorter system with contention buffering", *International Journal of Reconfigurable Computing*, vol. 2011, 2011. [Online]. Available: http://dx.doi.org/10.1155/2011/963539

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

[5] M. Zuluada, P. Milder, M. Puschel, "Streaming sorting networks", *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 4, 2016. [Online]. Available: http://dx.doi.org/10.1145/2854150

[6] V. Sklyarov, I. Skliarova, A. Sudnitson, "Fast data sort based on searching networks with ring pipeline", *Elektronika ir Elektrotechnika*, vol. 22, no. 4, pp. 58–62, 2016. [Online]. Available: http://dx.doi.org/10.5755/j01.eie.22.4.15920

[7] Digilent, Inc., *Nexys 4™ FPGA Board Reference Manual*, 2016. [Online]. Available: https://reference.digilentinc.com/lib/exe/fetch.php?media=nexys:nexys4:nexys4_rm.pdf