INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 11(1):126-141, February 2016.

# Computing Sorted Subsets for Data Processing in Communicating Software/Hardware Control Systems

V. Sklyarov, I. Skliarova, A. Rjabov, A. Sudnitson

**V. Sklyarov*, I. Skliarova**
University of Aveiro, Dept. of Electronics, Telecommunications and Informatics/IEETA
Campus Universitário de Santiago, 3810-193, Aveiro, Portugal
*Corresponding author: skl@ua.pt

**A. Rjabov, A. Sudnitson**
Tallinn University of Technology, Dept. of Computer Engineering
Akadeemia tee 15A, 12618, Tallinn, Estonia

**Abstract:** Computing and filtering sorted subsets are frequently required in statistical data manipulation and control applications. The main objective is to extract subsets from large data sets in accordance with some criteria, for example, with the maximum and/or the minimum values in the entire set or within the predefined constraints. The paper suggests a new computation method enabling the indicated above problem to be solved in all programmable systems-on-chip from the Xilinx Zynq family that combine a dual-core Cortex-A9 processing unit and programmable logic linked by high-performance interfaces. The method involves highly parallel sorting networks and run-time filtering. The computations are done in communicating software, running in the processing unit, and hardware, implemented in the programmable logic. Practical applications of the proposed technique are also shown. The results of implementation and experiments clearly demonstrate significant speed-up of the developed software/hardware system comparing to alternative software implementations.
**Keywords:** computing sorted subsets, communicating hardware/software systems, filtering, sorting networks, control applications.

## 1 Introduction

Many electronic, environmental, medical, and biological control applications need to process data streams produced by sensors and measure external parameters within given upper and lower bounds (thresholds) [1]. Let us consider some examples. Applying the technique [2] in real-time applications requires knowledge acquisition from controlled systems (e.g. plant). For example, signals from sensors may be filtered and analyzed to prevent error conditions (see [2] for additional details). To provide more exact and reliable conclusion, combination of different values need to be extracted, ordered, and analyzed. Similar tasks appear in monitoring thermal radiation from volcanic products [3], filtering and integration of information from a variety of different sources in medical applications [4] and in other practical applications described in [5]. Since many control systems are real-time, performance is important and hardware accelerators may provide significant assistance for software. A similar data processing is applicable to data mining algorithms, such as [6].

Let us consider control systems that collect, filter and analyze data produced by some measurements. We will describe below such computations that permit:

- the maximum and/or minimum sorted subsets to be extracted (the maximum/minimum sorted subset of size $L_{max}/L_{min}$ contains $L_{max}/L_{min}$ data items with maximum/minimum values from a given set);
- the maximum and/or minimum sorted subsets to be found within the given upper $B_u$ and lower $B_l$ bounds.

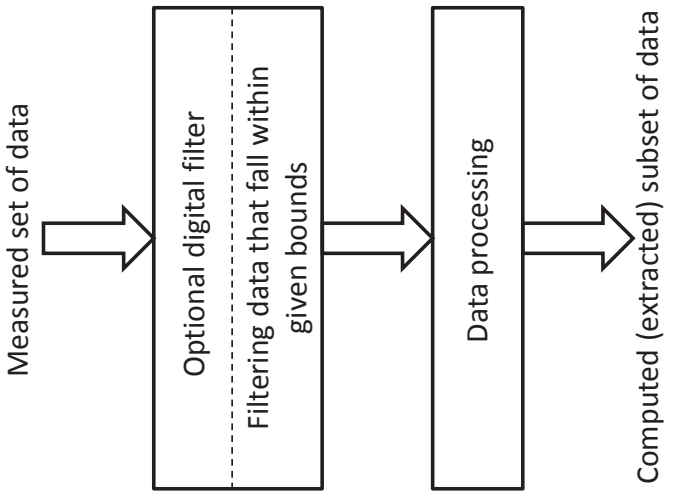


Figure 1: General architecture of data processing

The problem can be solved as it is shown in Fig. 1.

There are two blocks in Fig. 1. Measured data items are handled in such a way that the maximum and/or minimum subsets with $L_{max}$ and/or $L_{min}$ items are extracted by the data processing block. Input data may optionally be filtered allowing only items (such as D) that fall within pre-given constraints (e.g. $B_l \leq D \leq B_u$ or $B_l < D < B_u$) to be processed.

The paper suggests a method and high-performance implementation of architecture in Fig. 1 in all programmable systems-on-chip (APSoC) from the Xilinx Zynq-7000 family [7] that are recently developed field-configurable devices integrating the most advanced programmable logic (PL) and a widely used processing system (PS) based on the dual-core ARM® Cortex™ MP-Core™. The available interfaces between the PS and PL are supported by ready-to-use intellectual property (IP) cores. These, combined with numerous architectural and technological advances, have enabled APSoCs to open a new era in the development of highly optimized computational systems for a vast variety of practical applications, including high-performance computing, data, signal and image processing, control, and many others. The main target of APSoCs is integration in the developed systems of software and hardware components assuming that such integration enables characteristics (most often performance) of the system to be improved. The complexity of hardware only solutions is frequently limited by the available resources in the PL. Software/hardware solutions can be very complex and they are appropriate for control applications, such as that are described, for example, in [2,4]. The most close related work can be found in [5,8] where the importance of the considered problem is underlined, but the methods that allow the problem to be solved are different and the proposed below methods permit better results to be achieved.

The remainder of the paper is organized in eight sections. Section 2 presents the proposed software/hardware architecture. Section 3 describes a novel method allowing the maximum and minimum sorted subsets for a given set of data items to be computed. Section 4 suggests a run-time filtering method. Section 5 is dedicated to on-chip communication mechanisms linking software and hardware components. Section 6 shows how large subsets (for which hardware resources are not sufficient) can be computed and discusses additional capabilities such as extracting only the maximum or only the minimum subsets. Section 7 demonstrates potential practical application (from the areas of control and data mining). Implementations in Zynq microchips and the results of thorough evaluation and comparison of software only and software/hardware solutions with explicit indication of the achieved acceleration are discussed in section 8. Section 9 concludes the paper.

## 2 Software/Hardware Architecture

Fig. 2 presents the proposed software/hardware architecture.

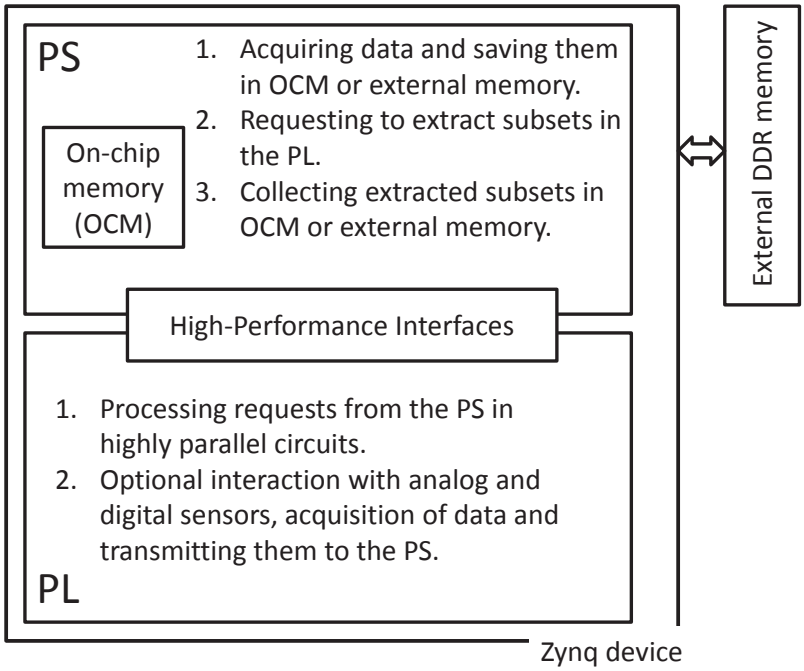


Figure 2: The proposed software/hardware architecture

The PS collects data, that may be acquired from different sources (such as from a host PC or from sensors connected to a Zynq device), and stores them in on-chip or external memory. The PL processes requests from the PS, that is, reads data from memories, and rapidly extracts the maximum and/or minimum subsets. Both parts, that are the PS and PL, may function in parallel and any request can be seen as a macroinstruction executed in the PL concurrently with other potential instructions in the PS.

It is shown in [9] that for transferring a small number of data items between the PS and the PL on-chip general-purpose ports (GPP) can be used more efficiently than other available interfaces. Thus, requests from the PS to the PL are formed through GPP where the PS is the master and the PL is a slave. It is also shown in [7, 9] that large volumes of data can be more efficiently transferred from/to memories to/from the PL through high performance (HP) interfaces: High-Performance Advanced eXtensible Interface (AXI HP) and AXI Accelerator Coherency Port (AXI ACP). In all our designs memories are slaves and either the PL or the processor in the PS is the master. To increase performance, data from memories may be requested to be cacheable.

## 3 Computing Sorted Subsets

Let set S containing N M-bit data items be given. The maximum subset contains $L_{max}$ largest items in S and the minimum subset contains $L_{min}$ smallest items in S ($L_{max} \leq N$ and $L_{min} \leq N$). We mainly consider such tasks for which $L_{max} << N$ and $L_{min} << N$ which are more common for practical applications. Since N may have very large value (millions of items) it cannot be processed completely in hardware due to the unavailability of sufficient resources. It is shown in [10, 11] that even for relatively complex Field-Programmable Gate Arrays (FPGAs) the size

N is limited. For example, for even-odd merge and bitonic merge networks [12] N cannot exceed a few hundreds of 32-bit items even for very advanced FPGAs (such as the largest devices from the Xilinx Virtex-7 family). In Zynq devices implementing circuits from [12] the maximum value of N does not exceed 128 32-bit items. Iterative even-odd transition networks from [11] permit significantly larger number of items (exceeding thousands of 32-bit items) to be processed and they will be used for computing sorted subsets in hardware. However, in practical cases the given sets anyhow cannot be entirely processed and computing the maximum and/or minimum sorted subsets needs to be done sequentially, nevertheless handling many items in parallel. Fig. 3 depicts the proposed architecture that enables the considered problem to be solved.

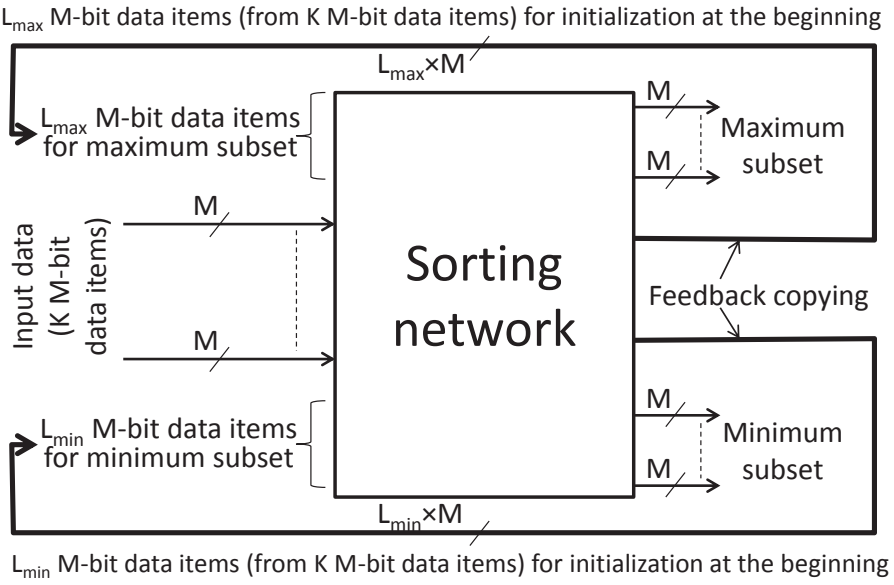


Figure 3: Computing the maximum and the minimum sorted subsets

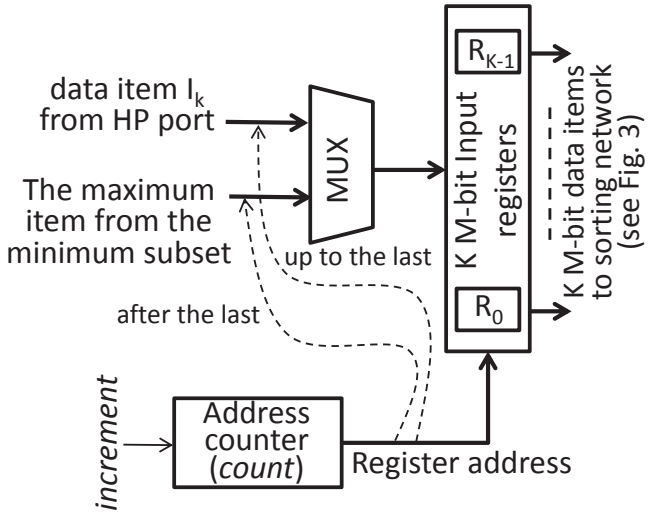


Figure 4: Processing the last (possibly incomplete) subset

Let us divide the given set S into Q = ⌈N/K⌉ subsets, all of which contain exactly K M-bit

items except the last one, which may have less than K M-bit items. Computing subsets is done incrementally in Q steps (we assume below that $K \leq N$).

At the first step, the first K M-bit data items are sorted in the network [11] which processes $L_{max}$+K+$L_{min}$ data items but comparators linking the upper part (handling $L_{max}$ M-bit data items) and the lower part (handling $L_{min}$ M-bit data items) are deactivated (i.e. the links with the upper and bottom parts are broken). So, sorting is done only in the middle part handling K M-bit items. As soon as the sorting is completed, the maximum subset is copied to the upper part of the network and the minimum subset is copied to the lower part of the network (see Fig. 3).

From the second step, all the comparators are properly linked, i.e. the network from [11] handles $L_{max}$+K+$L_{min}$ items, but the feedback copying (see the first step and Fig. 3) is disabled. Now for each new K M-bit items the maximum and the minimum sorted subsets are appropriately corrected, i.e. new items may be appended.

At the last step, the number of incoming items may be less than K. Fig. 4 explains how the maximum and minimum subsets are corrected for the last possibly incomplete subset of items. There is an additional MUX in Fig. 4, which supplies data items from a HP port (linking the PL with memory) until the received item is not the last. As soon as the last item is read from memory, the next items (until K) are taken as the maximum value from the minimum subset (see the lower subset in Fig. 3). Clearly, such an item cannot be moved again to the minimum subset and the last sorting step is executed similarly to the previous steps.

Let us look at the example shown in Fig. 5 for which: N = 21, K = 8, $L_{max} = L_{min} =$ 4, and S = 26,37,11,19,3,7,99,56,29,37,22,99,1,55,39,47,12,45,83,5,18. The set S is divided into the following three subsets: A = 26,37,11,19,3,7,99,56, B = 29,37,22,99,1,55,39,47, and C = 12,45,83,5,18.

Note that the last subset C contains only 5 elements and is incomplete. Symbol U in Fig. 5 indicates undefined value. The iterative sorting network is exactly the same as in [11]. Any comparator is shown in Knuth notation [13] and it converts two-item inputs in two-item outputs in such a way that the upper value is greater than or equal to the lower value. The maximum number of iterations for sorting is K/2 [14] and this number is almost always smaller because the method [11] terminates subsequent iterations as soon as all items are sorted. There are 3 steps in Fig. 5. At the first step, K (K=8) items are sorted and copied to the maximum and minimum subsets.

Two comparators are disabled in accordance with the explanations given above (breaking links of the middle section in the sorted network with the upper and the lower sections). At the second step, all the network comparators are enabled and $L_{max}$+K+$L_{min}$ items are sorted by the iterative network with feedback register (FR). All necessary details can be found in [11]. It is easy to show that the maximum number of iterations is $\lceil(\max(L_{max},L_{min})+K)/2\rceil$ and much like the previous case this number is almost always smaller [11]. At the last (third) step, the incomplete subset C is extended to K items by copying the maximum value (11) from the minimum subset 11,7,3,1 to the positions of missing data (see Fig. 5). After sorting $L_{max}$+K+$L_{min}$ items at the step 3 the final result is produced.

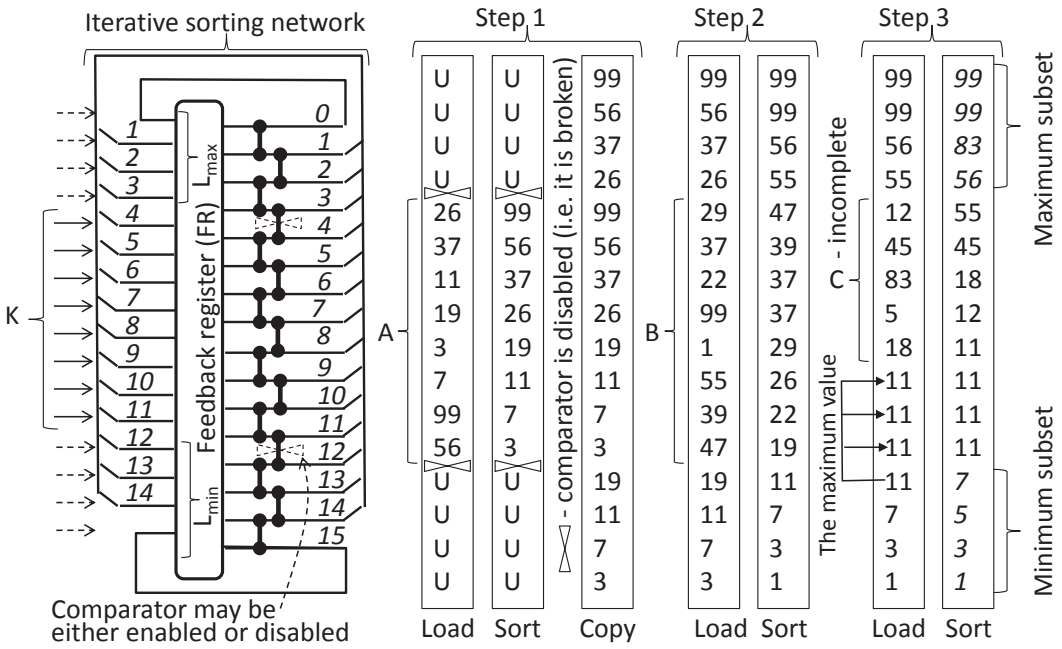


Figure 5: An example (computing subsets)

## 4 Filtering

Let $B_u$ and $B_l$ be predefined upper ($B_u$) and lower ($B_l$) bounds for the given set S. We would like to use the circuit in Fig. 3 only for such data items D that fall within the bounds $B_u$ and $B_l$, i.e. $B_l \leq D \leq B_u$ (or, possibly, $B_l < D < B_u$). Fig. 6 depicts the proposed architecture that enables data items to be filtered at run-time (i.e. during the data exchange between the PS and PL). There is an additional block on the upper input of the MUX (see also Fig. 4), which takes a data item $I_k$ from a HP port and executes the operation indicated on the right-hand part of Fig. 6. If the counter is incremented, then a new register is chosen to store $I_k$. Otherwise, the signal WE (write enable) is passive and a new item with a value that is out of the bounds $B_u$ and $B_l$ is not recorded in the registers.

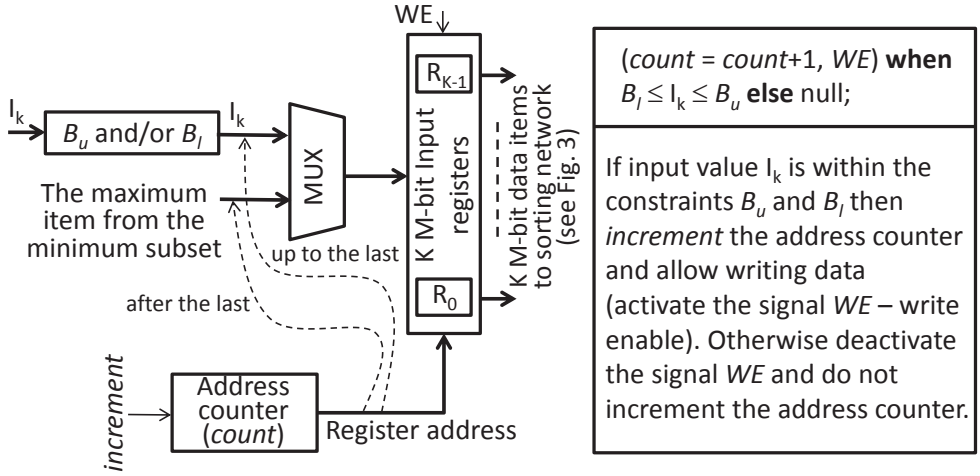


Figure 6: Digital filter

Let us look at the same example in Fig. 5 for which we choose $B_u = 90$ and $B_l = 10$ (see Fig. 7). At the first step incoming data items have preliminary been filtered, the values 99, 7, and 3 have been removed (because they are either greater than $B_u = 90$ or less than $B_l = 10$), and the subset A with 8 items is built from 11 first elements of the set S. At the second (last) step, the values 99, 1, and 5 have been removed, and the subset B = 55,39,47,12,45,83,18 is built from the remaining allowed elements of the set S. Since there are 7 items in B and K = 8, this subset is incomplete.

As can be seen from Fig. 7, two steps are sufficient to extract the maximum and the minimum subsets from the filtered set S. Similarly, filtering and computing sorted subsets can be done for very large data sets.

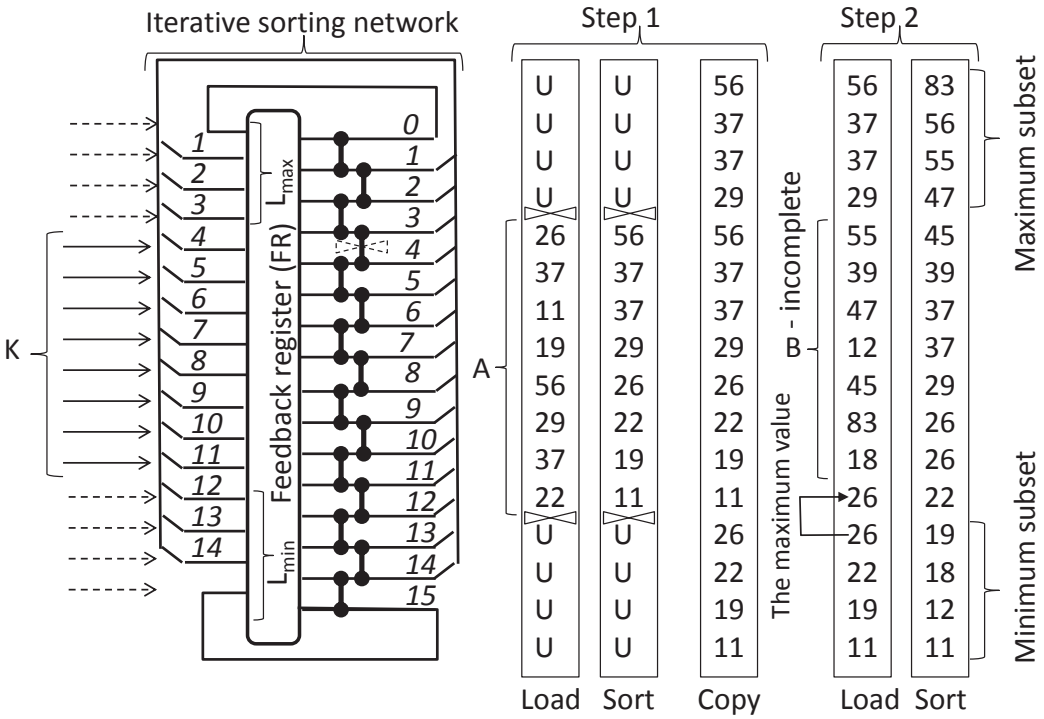


Figure 7: An example (filtering and computing subsets)

Clearly, the described above operations can be done in software. For example, C function qsort permits large data sets to be sorted. After that extracting the maximum and minimum subsets may easily be done. Filtering can be provided much like it is shown in Fig. 6 eliminating items that do not fall within the predefined constraints. However, for many practical applications performance of the described above operations is important. To evaluate software/hardware solutions three different components need to be taken into account (see Fig. 8): 1) software part; 2) hardware part; and 3) the circuits that provide for data exchange between software and hardware. Numerous experiments were done in [15] to compare such solutions with software only systems. One example in [15] enables sorting blocks of data composed of 320 32-bit items in the PL that are further merged in the PS (see Fig. 8). From 512,288 to 4,194,304 of 32-bit data items were randomly generated in the PS (i.e. the size of data varies from 2MB to 16MB) and then sorted in software with the aid of the function qsort and in the software/hardware system (see Fig. 8). The actual performance improvement was by a factor of about 2.5. It was shown in [15] that hardware circuits in the PL are significantly faster than software in the PS. In this paper we evaluate and compare software/hardware and software only solutions taking into account all the

involved communication overheads that were measured in [15]. We will mainly use AXI ACP [7] which provides one of the fastest interfaces for exchange of large data sets between the PS and PL [7, 9, 15]. The number of data items transferred from the PS/memory to the PL is the same as in [15]. However, the number of data items transferred from the PL to the PS/memory is significantly smaller enabling much better acceleration to be achieved.

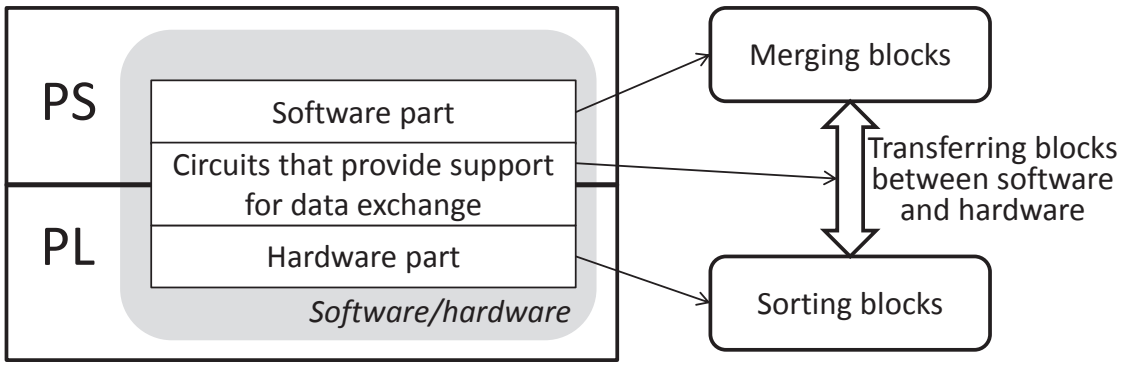


Figure 8: An example of a software/hardware system

## 5 Communication of Software and Hardware

Fig. 9 shows how communication is organized between software and hardware. It is done similarly to [8,15], but the proposed in this paper processing is different. The developed hardware in the PL is divided in two parts: application-specific (that is filtering and computing subsets) and communication-specific processing. The latter is studied in [15] and provides support for data exchange with storage of the PL that is either block RAM or registers built from flip-flops of configurable logic blocks.

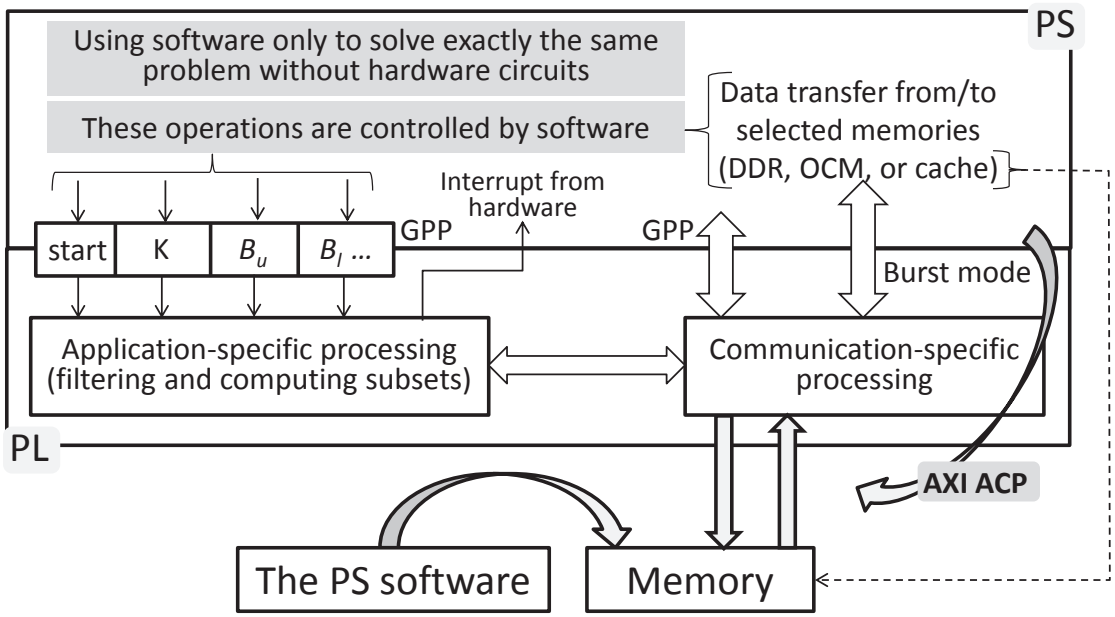


Figure 9: Communication between software and hardware components

Data are transmitted in blocks of 32/64-bit items (i.e. either M=32 or M=64) and the fastest burst mode is applied. Input data items $I_k$ (k=0,1,. . . ,K-1) are processed by the described above circuits (see Fig. 3, 4, 6). Note that GPP do not allow burst mode to be applied but are very appropriate for transferring small number of signals that may be used for control in the PL and for some additional details. In our system they are:

1) start requiring data processing in the PL to be initiated;

2) K (see Fig. 3);

3) $B_u$ and $B_l$ (see Fig. 6);

4) additional signals, namely source address, destination address and sizes of data to be transferred from the PS to the PL and vice versa.

Fig. 10 demonstrates a component diagram for reading the initial large volume data from the PS and for transferring the results (i.e. the computed maximum and minimum subsets) from the PL to the PS. We found that in such interactions between the PS and PL the best way is to use HP ports to read data from the PS (memories) and transfer them to the PL and to write data from the PL to the PS (memories). Since memory controllers belong to the PS we can talk about data transfers between the PS and PL. Exchange of data in both directions is done in burst mode supported by a burst reader and a burst writer described in [8, 15]. Both processing ($T_h$) and communication ($T_c$) times are measured and taken into account.

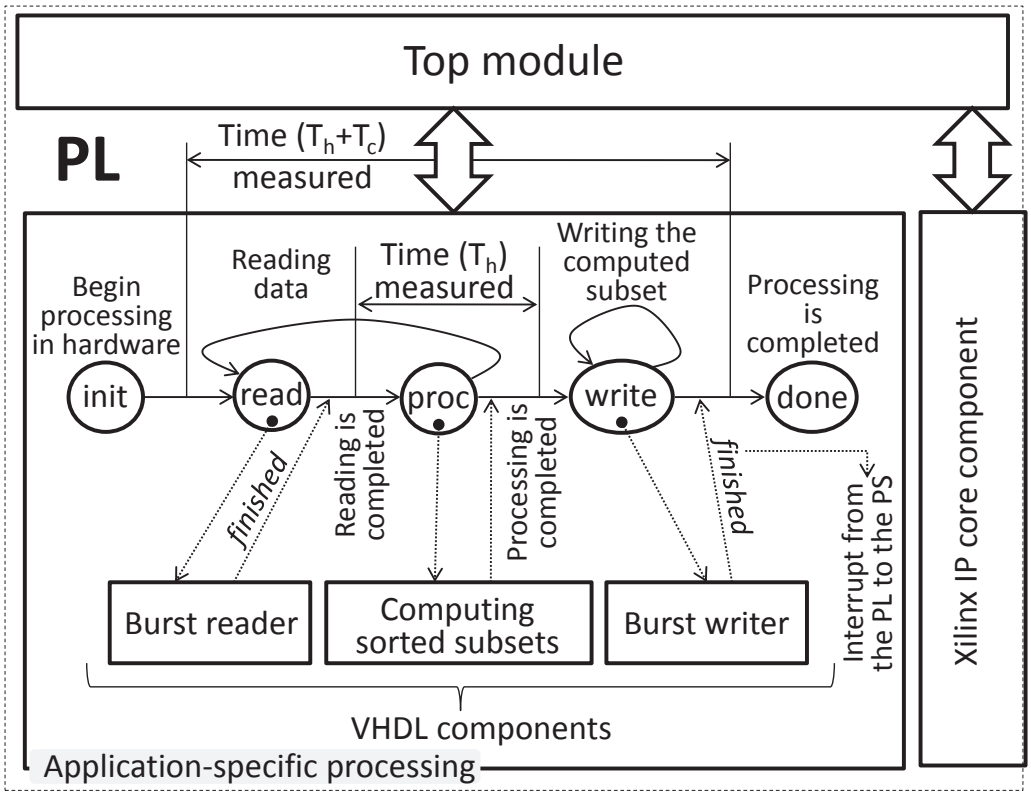


Figure 10: Operations in hardware components

## 6 Computing Large Subsets and Additional Capabilities

For some practical applications the maximum and/or minimum subsets may be large and the available hardware resources become insufficient to implement the circuits in Fig. 3.

The arising problem can be solved using the following technique. Let $l_{max}$ and $l_{min}$ be constraints for the upper and bottom parts of the sorting network in Fig. 3, i.e. circuits with larger values (than $l_{max}$ and $l_{min}$) cannot be implemented due to the lack of hardware resources or for some other reasons. Let the parameters for the maximum and minimum subsets be greater than $l_{max}$ and $l_{min}$, i.e. $L_{max} > l_{max}$ and $L_{min} > l_{min}$. In such case the maximum and minimum subsets can be computed incrementally [8] as follows:

1. At the first iteration the maximum subset containing $l_{max}$ items and the minimum subset containing $l_{min}$ items are computed. The subsets are transferred to the PS (to memories). The PS removes the minimum value from the maximum subset and the maximum value from the minimum subset. Such correction avoids loss of repeated items at subsequent steps. Indeed, the minimum value from the maximum subset (the maximum value from the minimum subset) can appear for subsets to be subsequently constructed in point 3 below and they will be lost because of filtering (see point 3).
2. The minimum value from the corrected in the PS maximum subset is assigned to $B_u$. The maximum value from the corrected in the PS minimum subset is assigned to $B_l$. The values $B_u$ and $B_l$ are supplied to the PL through GPP.
3. The same data items (from memory), as in point 1 above, are preliminary filtered (see Fig. 6) in such a way that only items that are less than $B_u$ and greater than $B_l$ are allowed to be processed, i.e. computing sorted subsets can be done only for the filtered data items. Thus, the second part of the maximum and minimum subsets will be computed and appended (in the PS) to the previously computed subsets (such as subsets from point 1). Note, that the method for processing incomplete subsets (see Fig. 4) may need to be applied for the last iteration.
4. The points 2 and 3 above are repeated until the maximum subset with $L_{max}$ items and the minimum subset with $L_{min}$ items are computed.

Note, that if the number of repeated items is greater than or equal to $l_{max}/l_{min}$, then the method above may generate infinite loops [8]. This situation can easily be recognized. Indeed, if after corrections in point 1 above any new subset becomes empty then an infinite loop will be created. In such case we can use another method based on software/hardware sorters from [9]. In section 8 we will present the results of experiments for such sorters.

For some practical cases only the maximum or the minimum subsets need to be extracted. This task can be solved easier than in Fig. 3 with the aid of the circuit shown in Fig. 11 (for computing only the maximum subset).

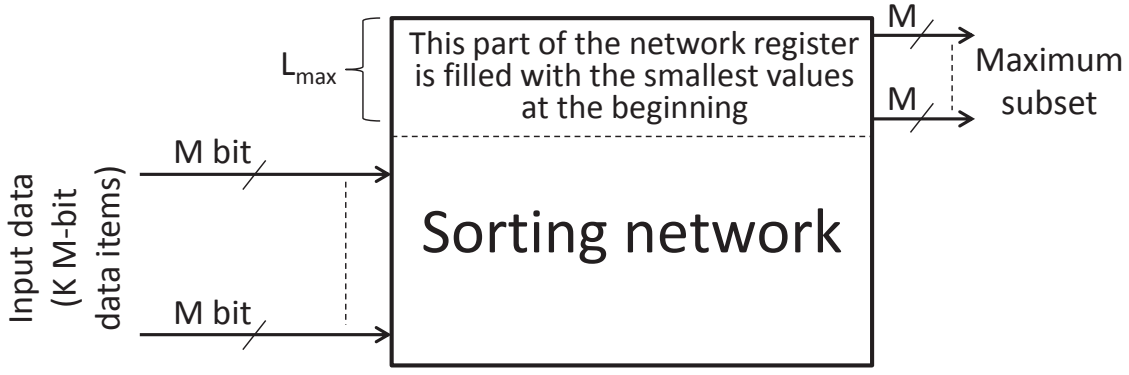


Figure 11: Computing the maximum subset for a given set

At initialization stage $L_{max}$ M-bit words of the FR (see Fig. 5) are filled in with the smallest

possible value (such as zero or the minimal value for M-bit data items). After that the processing is executed as before (see section 3) and finally the maximum subset will be computed. For computing the minimum subsets the bottom part of Fig. 3 is filled in with the largest possible value (such as the maximum value for M-bit data items).

## 7 Practical Applications

Let us consider practical applications from the scope of control. We have already mentioned in section 1 that applying the technique [2] in real-time systems requires knowledge acquisition from controlled devices. The data may be compared with the previously collected data that are kept in databases for similar control scenarios. The results of comparison can be analyzed and used to modify the algorithms allowing control operations to be optimized, undesirable (or error prone) situations to be avoided, etc. Let us look at Fig. 12 where software collects important data from a controlled system, such as changes in temperature, deviation of positions, offsets, etc. The collected data are optionally filtered and their subsets (maximum, minimum, or both) are computed (see the bottom part of Fig. 12). Data from previous scenarios for analogous conditions are extracted from the database and they are also optionally filtered and similar subsets are computed (see the upper part of Fig. 12).

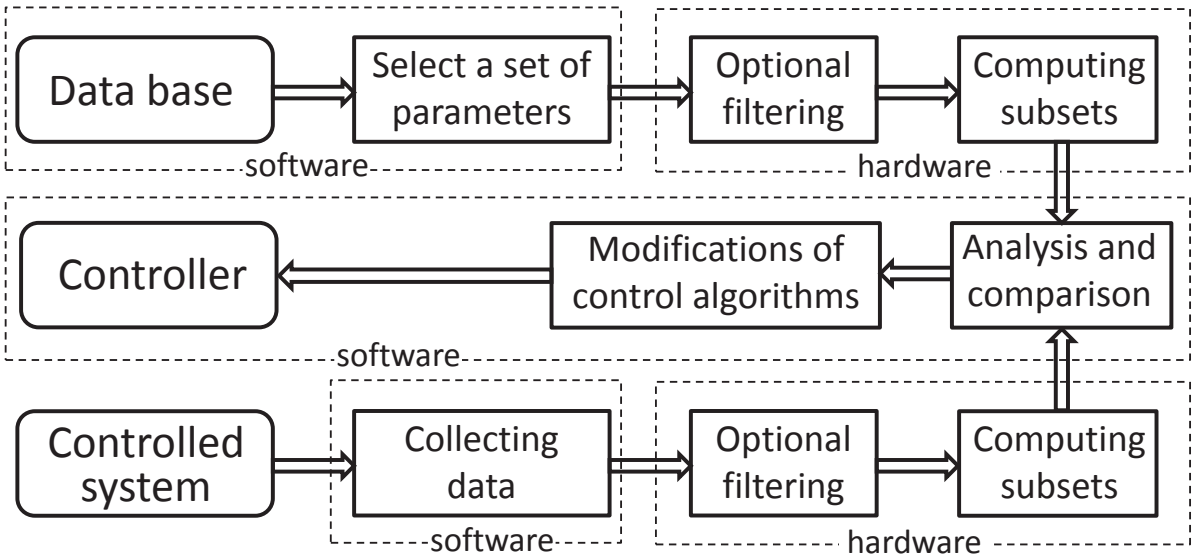


Figure 12: An example of control application

Data from the controlled system (see the bottom part of Fig. 12) and from the database (see the upper part of Fig. 12) are analyzed. For example, average maximum values are checked. The results of analysis may be used to modify control algorithms much like it is done in [9, 16]. For example, modules of controllers from [9] can be replaced to optimize execution of relevant operations.

Another group of potential applications is from the scope of statistical data manipulation such as data mining. To describe one of the problems from this area informally let us consider an example [6] with analogy to a shopping card. A basket is the set of items purchased at one time. A frequent item is an item that often occurs in a database. A frequent set of items often occur together in the same basket. A researcher can request a particular support value and find the items which occur together in a basket either a maximum or a minimum number of times

within the database [6]. Similar problems appear to determine frequent inquiries at the Internet, customer transactions, credit card purchases, etc. producing very large volumes of data in the span of a day [6]. Computing sets of the most frequent or the less frequent items in large data sets permits the relevant data mining algorithms to be simplified and accelerated. Sorting of subsets is involved in many known algorithms from this area e.g. [17–19] and the results of the paper may provide a valuable assistance.

## 8 Implementation, Experiments, and Comparisons

Much like [8] we have used a multi-level computing system [9]. Initial data are either generated randomly in software of the PS with the aid of C language rand or prepared in the host PC. In the last case data may be generated by some functions or copied from available benchmarks. Computing subsets in software/hardware systems is mainly done in Zynq APSoC xc7z020 housed on ZedBoard [20] with the aid of the described above software/hardware architectures (see Fig. 2-4, 6, 9, 10). Computing subsets in software only sorters is completely done in software of the PS calling C language qsort function which sorts data and after that the maximum and/or the minimum subsets are extracted. The results are verified in software running either in the PS or in the host PC. Functions for verification of the results are given in [9]. Verification time is not taken into account in the measurements below.

Synthesis and implementation of hardware modules were done in Xilinx Vivado 2015.2. Standalone software applications were created in C language and uploaded to the PS memory from the Xilinx Software Development Kit (SDK) using methods described in [9]. Interactions with APSoC are done through the SDK console window.

For all the experiments 64-bit AXI ACP port was used for transferring blocks between the PL and memories. The size of each block for burst mode is chosen to be 128 of 64-bit items. Two memories were tested: the OCM (for smaller number of data items) and external (on-board) DDR. The OCM is faster because it provides for 64-bit data transfers [7] but the size of this memory is limited to 256 KB. The available on ZedBoard 4 Gb DDR supports 32-bit data transfers.

The measurements were based on time units (returned by the function XTime_GetTime [21]) for $L_{max} = L_{min} = 128$, M=32, and K = 256 (see Fig. 3). The following operations have been executed: a) copying data to the selected memory in the PS; b) providing the necessary initialization for the function XTime_GetTime (i.e. the consumed time will be measured from this point); c) making the request, i.e. setting (through GPP) source address, destination address, the size of data to be copied, and start processing in the PL (optionally some other data, such as $B_u$ and $B_l$ for filtering, may be provided); d) copying data from the PS to PL and executing all the required operations in the PL; e) copying the computed subsets from the PL to PS; f) generating a hardware interrupt that is handled in the PS as a completion of the request (thus, the consumed time is measured at this point in the PS). Each unit returned by the function XTime_GetTime corresponds to 2 clock cycles of the PS [21]. The PS clock frequency is 666 MHz. Thus, any unit corresponds to approximately 3 ns. The PL clock frequency was set to 100 MHz.

Fig. 13 shows the time consumed for computing the maximum and minimum subsets for data sets with different sizes in KB (from 2 to 128). Since M=32 the number of processed words (N) is equal to the indicated size divided by 4.

Fig. 14 shows the acceleration of the software/hardware system comparing to the software only system. Note that Fig. 13, 14 give diagrams for the OCM. If DDR memory is used then communication overheads are slightly increased but acceleration in the software/hardware system comparing to the software only system is again significant.

Let us compare the results with [5, 8]. The number of data items in the proposed solutions is larger than in [5] and can easily be additionally increased. For similar data sets the achieved acceleration is better than in [8] thanks to additional optimization of the proposed circuits.

We also implemented and tested the proposed circuits in a more advanced prototyping board ZC706 [22] with Zynq microchip xc7z045. Data were taken from DDR memory and the maximum and minimum subsets were extracted with K data items where K varied from 256 to 1,024 (as before M = 32, N is equal to 256 KB). The consumed time varies from 1,850 $\mu s$ for $L_{min}$ = $L_{max}$ = 256 to 7,200 $\mu s$ for $L_{min}$ = $L_{max}$ = 1,024. Thus, the proposed solutions can be used for solving significantly more complicated problems that cannot be solved, in particular, with the aid of the methods [5]. If only the maximum or only the minimum subsets have to be computed the acceleration is slightly increased (although it is almost the same) and the occupied hardware resources are reduced.

The proposed filtering (see Fig. 6) does not consume any additional time because it is combined with data transfers. So, we can say that the time is included in communication overheads and the latter were taken into account in all measurements. It should be noted that filtering is not described in [5, 8].

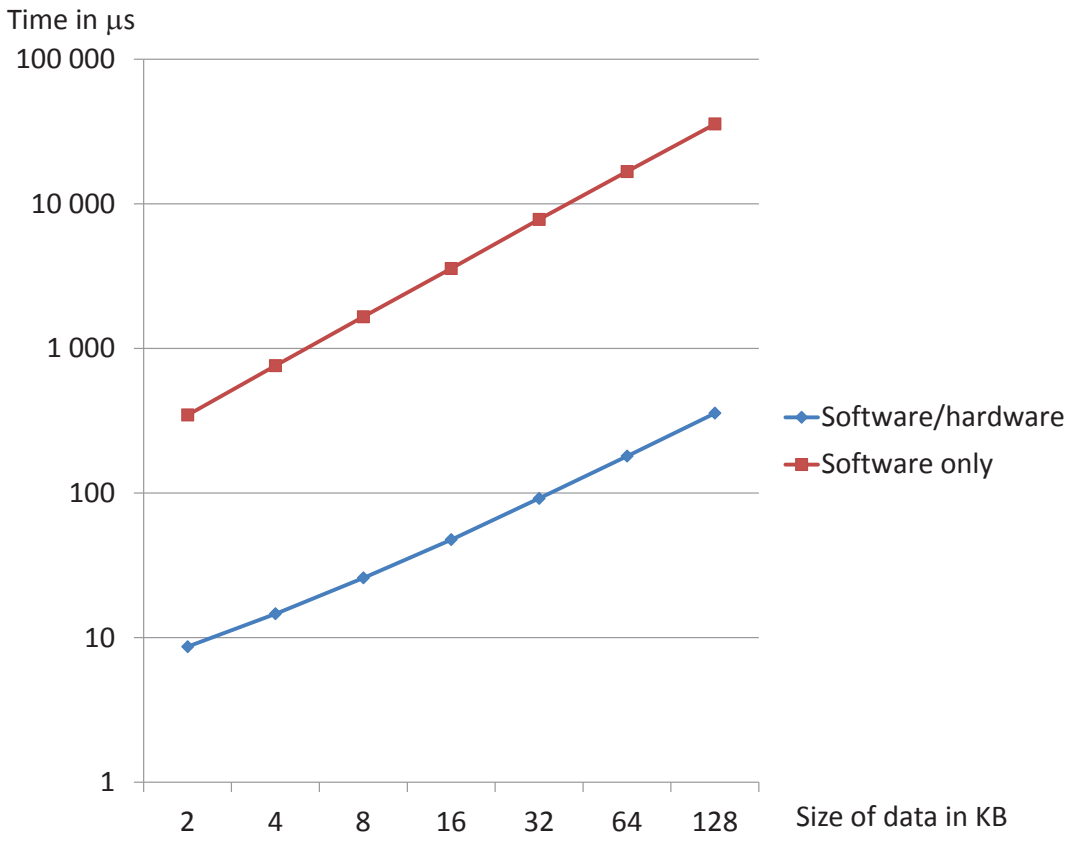


Figure 13: Computing time in software only and software/hardware systems

If the size of the requested subsets is increased in such a way that all data need to be read from memory several times then the results are the same as in [8] (see comments in [8] for additional details).

We found that parallel circuits that enable the maximum and minimum subsets to be ex-

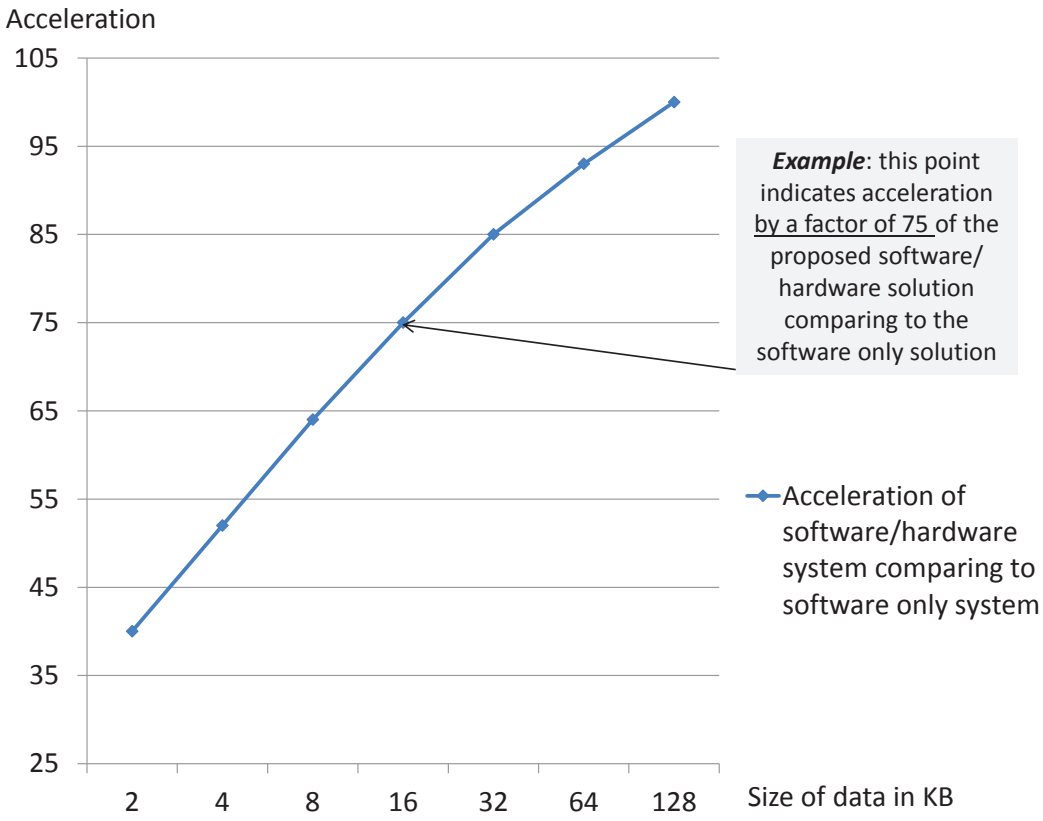


Figure 14: Acceleration of software/hardware system comparing to software only system

tracted in the ZedBoard [20] can be built up to K = 256. In this case additional hardware resources that enable data exchange between the PS (memories) and PL are available. Similar circuits for the ZC706 can be built up to K = 1,024. Note that if a block of data needs to be sorted in hardware then the number of processed data may be greater because in our case two blocks (each of which possesses K items) have to be handled in parallel and in case of data sorting [9] it is sufficient to handle just one block of data. Additional optimizations such as partial merging in hardware circuits permit the size K to be additionally increased. However, the processing time will also be increased.

## 9 Conclusion

The paper suggests methods for computing the maximum and minimum subsets that are extracted from large data sets in communicating software/hardware systems, namely in devices from the Xilinx Zynq family, which combine a high-performance processing system with advanced programmable logic. The extracted subsets may be filtered and this feature is useful for control applications. The proposed solutions are highly parallel permitting capabilities of programmable logic to be used very efficiently. All the proposed methods were implemented in commercial microchips, tested, evaluated, and compared with alternatives. The results of experiments have shown significant speed-up of the proposed software/hardware systems comparing to software only systems and to competitive hardware/software implementations. In particular, the size of

subsets was increased and additional tasks important for control applications were discussed and solved. Practical applications of the proposed technique for control applications and statistical data manipulation were also given.

## Acknowledgment

This research was supported by EU through European Regional Development Funds, the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research, ESF grant 9251, and Portuguese National Funds through FCT - Foundation for Science and Technology, in the context of the project PEst-OE/EEI/UI0127/2014.

## Bibliography

[1] Sklyarov, V.; Skliarova, I. (2013); Digital Hamming Weight and Distance Analyzers for Binary Vectors and Matrices, *Int. Journal of Innovative Computing, Information and Control*, 9(12): 4825-4849.

[2] Zmaranda, D.; Silaghi, H.; Gabor, G.; Vancea, C. (2012); Issues on applying knowledge-based techniques in real-time control systems, *International Journal of Computers Communications & Control*, 8(1): 166-175.

[3] Field, L.; Barnie, T.; Blundy, J.; Brooker, R. A.; Keir, D.; Lewi, E.; Saunders, K. (2012); Integrated field, satellite and petrological observations of the November 2010 eruption of Erta Ale, *Bulletin of Volcanology*, 74(10): 2251-2271.

[4] Zhang, W.; Thurow, K.; Stoll, R. (2014); A knowledge-based telemonitoring platform for application in remote healthcare, *International Journal of Computers Communications & Control*, 9(5): 644-654.

[5] Farmahini-Farahani, A.; Duwe, H. J.; Schulte, M. J.; Compton, K. (2013); Modular design of high-throughput, low-latency sorting units, *Computers, IEEE Transactions on*, 62(7): 1389-1402.

[6] Baker, Z. K.; Prasanna, V. K. (2006); An architecture for efficient hardware data mining using reconfigurable computing systems. In Field-Programmable Custom Computing Machines, *Proc. 14th Annual IEEE Symp. on Field-Programmable Custom Computing Machines - FCCM*, Napa, USA, 67-75.

[7] Xilinx, Inc.; Zynq-7000 All Programmable SoC Technical Reference Manual, available at http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

[8] Sklyarov, V.; Skliarova, I.; Rjabov, A.; Sudnitson, A. (2015); Zynq-based System for Extracting Sorted Subsets from Large Data Sets, *Journal of Microelectronics, Electronic Components and Materials*, 45(2): 142-152.

[9] Sklyarov, V.; Skliarova, I.; Silva, J.; Rjabov, A.; Sudnitson, A.; Cardoso, C. (2014); *Hardware/software co-design for programmable systems-on-chip*, TUT Press.

[10] Mueller, R.; Teubner, J.; Alonso, G.; (2012); Sorting networks on FPGAs, *The VLDB Journal—The International Journal on Very Large Data Bases*, 21(1), 1-23.

[11] Sklyarov, V.; Skliarova, I. (2014); High-performance implementation of regular and easily scalable sorting networks on an FPGA, *Microprocessors and Microsystems*, 38(5): 470-484.

[12] Baddar, S. W. A. H.; Batcher, K. E. (2012); *Designing sorting networks: A new paradigm*, Springer Science & Business Media.

[13] Knuth, D. E. (2011); *The art of computer programming: sorting and searching (Vol. 3)*, Addison-Wesley.

[14] Kipfer, P.; & Westermann, R. (2005); Improved GPU sorting, *GPU gems 2: programming techniques for high-performance graphics and general-purpose computation*, edited by M. Pharr, 733-746, available at http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter46.html.

[15] Silva, J.; Sklyarov, V.; Skliarova, I. (2015). Comparison of On-chip Communications in Zynq-7000 All Programmable Systems-on-Chip, *Embedded Systems Letters, IEEE*, 7(1): 31-34.

[16] Sklyarov, V.; Skliarova, I.; Barkalov, A.; Titarenko, L. (2014); *Synthesis and Optimization of FPGA-based Systems*, Springer.

[17] Sun, S. (2011); *Analysis and acceleration of data mining algorithms on high performance reconfigurable computing platforms*, Ph.D. thesis, Iowa State University, available at: http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1421&context=etd.

[18] Wu, X.; Kumar, V.; Quinlan, J. R. et al. (2008); Top 10 algorithms in data mining, *Knowledge and Information Systems*, 14(1): 1-37.

[19] Firdhous, M. (2012); Automating legal research through data mining, *Journal of Advanced Computer Science and Applications*, 1(6): 1-8.

[20] Avnet, Inc. (2014); *ZedBoard (ZynqTM Evaluation and Development) Hardware User's Guide, Version 2.2*, available at: http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[21] Xilinx, Inc.; *OS and Libraries Document Collection UG647*, available at: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/oslib_rm.pdf.

[22] Xilinx, Inc.; *ZC706, All Programmable SoC Evaluation Kit, UG961*, Available at: http://www.xilinx.com/support/documentation/boards_and_kits/zc706/2014_3/ug961-zc706-GSG.pdf.