International Journal of Innovative Computing, Information and Control
Volume **12**, Number **1**, February **2016**
 ISSN 1349-4198
pp. **139–160**

HIGH-PERFORMANCE INFORMATION PROCESSING IN DISTRIBUTED COMPUTING SYSTEMS

Valery Skliarov[1], Artjom Rjabov[2], Iouliia Skliarova[1] and Alexander Sudnitson[2]

[1]Department of Electronics, Telecommunications and Informatics
Institute of Electronics and Informatics Engineering of Aveiro
University of Aveiro
Aveiro 3810-193, Portugal
{ skl; iouliia }@ua.pt

[2]Department of Computer Engineering
Tallinn University of Technology
Tallinn 19086, Estonia
aleksander.sudnitson@ttu.ee

Received July 2015; revised December 2015

Abstract. *This paper explores distributed computing systems that may be used efficiently in information processing that is frequently needed in electronic, environmental, medical, and biological applications. Three major components of such systems are: 1) data acquisition and preprocessing; 2) transmitting the results of preprocessing to a higher level computing system that is a PC; and 3) post processing in higher level computing system (in the PC). Preprocessing can be done in highly parallel accelerators that are mapped to reconfigurable hardware. The core of an accelerator is a sorting/searching network that is implemented either in an FPGA or in a programmable system-on-chip (such as Zynq devices). Data is transmitted to a PC through a high-bandwidth PCI-express bus. The paper suggests novel solutions for sorting/searching networks that enable the number of data items that can be handled to be significantly increased compared to the best known alternatives, maintaining a very high processing speed that is either similar to, or higher than in the best known alternatives. Preprocessing can also include supplementary tasks, such as extracting the minimum/maximum sorted subsets, finding the most frequently occurring items, and filtering the data. A higher level computing system executes final operations, such as merging the blocks produced by the sorting networks, implementing higher level algorithms that use the results of preprocessing, statistical manipulation, analysis of existing and acquired sets, data mining. It is shown through numerous experiments that the proposed solutions are very effective and enable a more diverse range of problems to be solved with better performance.*
Keywords: High-performance computing systems, Information processing, Sorting, Searching, Merging, Reconfigurable hardware, PCI-express bus, Programmable systems-on-chip

1. **Introduction.** Sorting and searching procedures are needed in numerous computing systems [1]. They can be used efficiently for data extraction and ordering in information processing. Some common problems that they apply to are (see also Figure 1):

1. Extracting sorted maximum/minimum subsets from a given set;
2. Filtering data, i.e., extracting subsets with values that fall within given limits;
3. Dividing data items into subsets and finding the minimum/maximum/average values in each subset, or sorting each subset;

4. Finding the value that is repeated most often, or finding the set of n values that are repeated most often;
5. Removing all duplicated items from a given set;
6. Computing medians;
7. Solving the problems indicated in points 1-6 above for matrices (for rows/columns of the matrices).

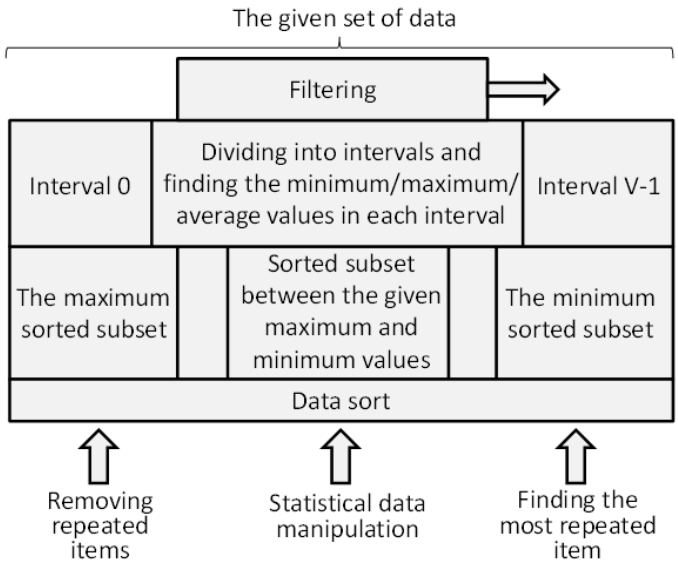

FIGURE 1. Common problems that are frequently solved in information processing systems

These problems are important because many electronic, environmental, medical, chemical, and biological applications need to process data streams produced by sensors and calculate certain parameters [2]. Let us consider some examples. Applying the technique [3] in real-time applications requires data acquisition from control systems such as in a manufacturing or process plant. Signals from sensors may need to be filtered and analyzed to prevent error conditions (see [3] for additional details). To provide a more precise and reliable conclusion, combinations of different values need to be extracted, ordered, and analyzed. Similar tasks arise in monitoring thermal radiation from volcanic eruptions [4], filtering and integrating information from a variety of sources in medical applications [5], in data mining [6], and so on. Since many control systems are real-time, performance is important and hardware accelerators can provide significant assistance for software.

The problems listed above can be solved as shown in Figure 2. Measured data items are handled in such a way that they are optionally filtered before various types of data processing algorithms are applied. Performance can be increased by employing broad parallelism and we suggest providing support for such parallelism in networks for sorting and searching.

Let us introduce a set of high-level operations, each of which is dedicated to one of the problems listed in points 1-6 above. The paper suggests methods for high-performance implementation of such operations in a distributed computing system with the architecture depicted in Figure 3.

Two basic subsystems (a pre-processor implemented in reconfigurable devices and a host PC) communicate through a high-bandwidth PCI-express (Peripheral Components Interface) bus. Two types of reconfigurable devices are studied: advanced field-programmable gate arrays (FPGAs), such as those from the Xilinx Virtex-7 family, and all-programmable

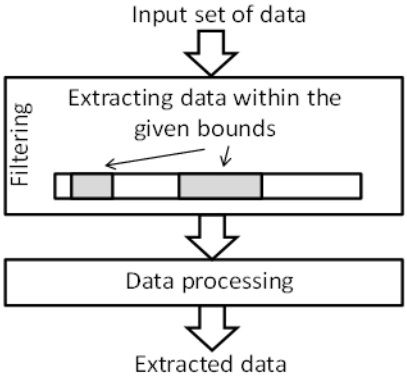

FIGURE 2. General architecture of data processing

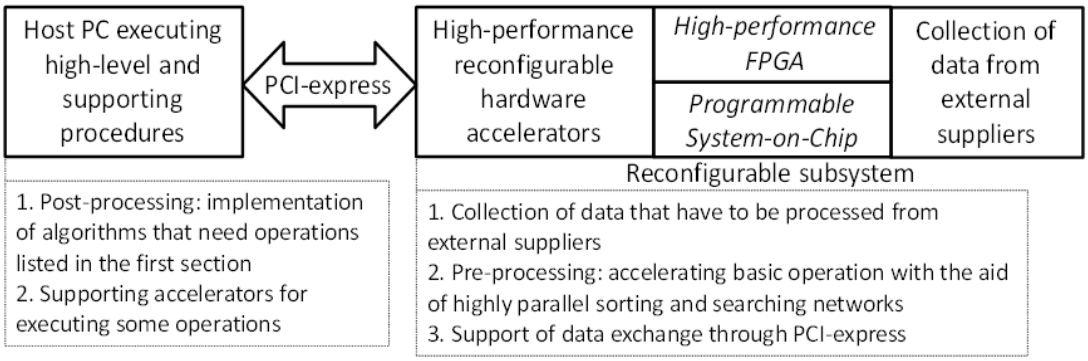

FIGURE 3. Architecture of a distributed computing system

systems-on-chip (APSoCs), such as those from the Xilinx Zynq-7000 family. In the first case, two-levels of processing are involved: accelerators implemented in reconfigurable hardware, and general-purpose software running in a PC. In the latter case, a third level of processing is added that is application-specific software running in a multi-core processing unit embedded to APSoC. The basic tasks of the host PC and the reconfigurable subsystem are listed in Figure 3.

The main contributions of the paper can be summarized as follows:

1) Selecting widely reusable operations for various types of information processing, and developing a methodology for using such operations in engineering applications;
2) Highly parallel networks for various sorting and searching algorithms and their thorough evaluation;
3) Multi-level implementation of the selected operations in general-purpose and application-specific software, and in reconfigurable hardware;
4) Experimental evaluation of the proposed technique in two advanced prototyping boards – the ZC706 [7] and the VC707 [8].

The remainder of the paper contains 6 sections. Section 2 analyzes related work. Section 3 identifies potential practical applications of the results and gives a number of real world examples from different areas. Section 4 describes highly parallel networks for sorting and searching. Section 5 explores software/hardware co-design. Section 6 presents the results of experiments and comparisons. The conclusion is given in Section 7.

2. **Related Work.** The majority of known methods and designs that are relevant to this paper are in three main areas: 1) high-performance distributed systems that include

a general-purpose computer (such as a PC) and a reconfigurable subsystem interacting through a high-bandwidth PCI-express bus; 2) sorting and searching using highly parallel networks; 3) software/hardware co-design targeted to combine reconfigurable hardware with general-purpose and application-specific software. The following subsections discuss the related work that has been done in each area separately.

2.1. **High-performance distributed systems.** High-performance distributed systems are used in many areas such as in medical equipment, aerospace, transportation vehicles, intelligent highways, defense, robotics, process control, factory automation, and building and environmental management [9]. A number of such systems for different application areas are described in [10-12]. Let us consider one of them from [12]. Typical adaptive cruise control and collision avoidance systems receive periodic inputs from sensors such as radar, lidar (light identification detection and ranging), and cameras, and then process the data to extract the necessary information. The system then executes a control algorithm to decide how much acceleration or deceleration is required, and sends commands to actuators to execute the appropriate actions. Since this is time-critical functionality, the end-to-end latency from sensing to actuation must be bounded.

The application domains define different requirements for systems. Independently of the domains, the majority of applications need data processing that may be organized in different ways. For example, a researcher may require support for finding data items that occur together in a certain situation, either a maximum or a minimum number of times. Such problems arise in determining the frequency of inquiries over the Internet, for customer transactions such as credit card purchases, which typically produce very large volumes of data in the course of a day [6]. Data processing is involved in software systems [13], priority buffering in scheduling algorithms [10], information retrieval [14], extracting data from sensors within predefined ranges [2], video processing [15], knowledge acquisition from controlled environments [3], and so on. Satisfying the real-time requirements for these applications can be achieved in on-chip devices that combine a multi-core processing system (PS) running multi-thread software with programmable logic (PL) that can be used to implement hardware accelerators. For example, devices from the Xilinx Zynq-7000 family of APSoC have already been successfully used in a number of engineering designs [11,12]. The Zynq APSoC combines the dual-core ARM®Cortex™-A9 central processing unit with the PL appended with on-chip memories (OCM), high-performance (HP) interfaces, a rich set of input/output peripherals, and a number of embedded to the PL components, such as digital signal processing (DSP) slices. APSoC devices enable complete solutions to be implemented on a single microchip running software that may be enhanced with easily customizable hardware. Various advantages of the APSoC platform are summarized in [16,17]. Interactions between the ARM-based PS and PL are supported by nine on-chip Advanced eXtensible Interfaces (AXI): four 32-bit general-purpose (GP) ports; four 32/64-bit HP ports, and one 64-bit accelerator coherency port (ACP) [15]. There are a number of prototyping systems available, some of which (e.g., [7]) allow additional data exchange with higher level computers such as PCs through PCI-express. On-chip interfaces [18] enable hardware accelerators and other circuits (supporting, for example, communications with sensors and actuators) in the PL to be linked with multi-core software running in the PS. The PCI-express bus permits multilevel systems to be developed that combine software running in a general-purpose computer (e.g., PC), software running in the PS, and hardware implemented in the PL.

Satisfying real-time requirements is critical in many of the systems referenced above and this can be a problem for the software-only systems that, perhaps, are the most frequently applied nowadays [11,13]. Hardware-only systems are also widely used in a number of

areas (e.g., [12]). It is concluded in [12] that the most sensible approach is for the data intensive portions of an application to be implemented in hardware, thus providing a high degree of determinism and lower execution time, while the high-level decision making is implemented in software, supporting easy customization. It is shown in [19] that on-chip interactions between software and hardware may be seen as a bottleneck (even if HP ports are used) especially for applications that require the exchange of high volumes of data.

The architectures and functionalities of various PCI-express systems are described in [20]. A PCI connection has one or more data transmission lanes, each of which consists of two pairs of wires: one for receiving and one for sending data. The maximum theoretical bandwidth of a single lane is up to 2.5 Giga transfers per second (GT/s) in each direction simultaneously [21], which is the same as Gb per second except that some bits are lost as a result of interface overhead and consequently the theoretical bandwidth is reduced by approximately 20% [21,22]. The bandwidth of χ lanes is the bandwidth of one lane multiplied by χ.

There are many systems that involve high-performance on-chip communications and interactions with PC through a PCI-express bus. The distinctive feature of this paper is the study and evaluation of such systems for a particular area of data processing targeted to problems that were described in Section 1. We will show in the next section that there exist a very large number of potential practical applications for such processing.

2.2. **Sorting and searching networks.** Highly parallel networks for sorting and searching enable numerous operations to be executed simultaneously, which is very appropriate for FPGAs and APSoCs. Two of the most frequently investigated parallel sorters are based on sorting [23] and linear [24] networks. A sorting network is a set of vertical lines composed of comparators that can swap data to change their positions in the input multi-item vector. The data propagate through the lines from left to right to produce the sorted multi-item vector on the outputs of the rightmost vertical line. Three types of such networks have been studied: pure combinational (e.g., [23,25,26]), pipelined (e.g., [23,25,26]), and combined (partially combinational and partially sequential) (e.g., [27]). The linear networks, which are often referred to as linear sorters [24], take a sorted list and insert new incoming items in the proper positions. The method is the same as the insertion sort [1] that compares a new item with all the items in parallel, then inserts the new item at the appropriate position and shifts the existing elements in the entire multi-item vector. Additional capabilities of parallelization are demonstrated in the interleaved linear sorter system proposed in [24]. The main problem with this is that it is applicable only for small data sets (see, for example, the designs discussed in [24], which accommodate only tens of items).

The majority of sorting networks that are implemented in hardware use Batcher even-odd and bitonic mergers [28,29]. Other types are rarer (see for example the comb sort [30] in [31], the bubble and insertion sort in [23,25], and the even-odd transition sort in [32]). Research efforts are concentrated mainly on networks with a minimal depth or number of comparators and on co-design, rationally splitting the problem between software and hardware. The regularity of the circuits and interconnections are studied in [26,27] where networks with iteratively reusable components were proposed. We target our results towards FPGAs and APSoCs because they are regarded more and more as a universal platform incorporating many complex components that were used autonomously not so long ago. The majority of modern FPGAs contain embedded DSP slices and embedded multi-port memories, which are very appropriate for sorting. GPU (graphics processing unit) cores have also been placed inside an APSoC in recent devices [17], but even without

such cores, streaming SIMD (single instruction multiple data) applications can be supported with the existing programmable logic. Comparing FPGA-based implementations with alternative systems [33,34] clearly demonstrates the potential of reconfigurable hardware, which encourages further research in this area. FPGAs still operate at a lower clock frequency than non-configurable ASICs (application-specific integrated circuits) and ASSPs (application-specific standard products) and broad parallelism is evidently required to compete with potential alternatives. Thus, sorting and linear networks can be seen as very adequate models. Unfortunately, they have many limitations. Suppose N data items, each of size M bits, need to be sorted. The results of [23,25] show that sorting networks cannot be built for $N > 64$ ($M = 32$), even in the relatively advanced FPGA FX130T from the Xilinx Virtex-5 family because the hardware resources are not sufficient. When N is increased, the complexity of the networks (the number of comparators $C(N)$) grows rapidly (see Figure 1 in [26]). Besides, propagation delays through long combinational paths in FPGA networks are significant [26]. Such delays are caused not only by comparators, but also by multiplexers that have to be inserted even in partially regular circuits [27], and by interconnections.

It is shown in [26] that very regular even-odd transition networks with two sequentially reusable vertical lines of comparators are more practical because they operate at a higher clock frequency, provide sufficient throughput, and enable a significantly larger number of items to be processed in programmable logic.

2.3. **Multi-level software/hardware co-design.** Multi-level software/hardware co-design is a new kind of system design that combines on-chip hardware/software co-design and co-design at the level of PC interacting with the on-chip subsystem, such as that implemented in Zynq-7000 devices. To our knowledge just a few publications (such as [35,36]) briefly discuss such multi-level co-designs. On the other hand existing prototyping boards, such as [7] permit such designs to be evaluated. Besides, they are valuable for numerous practical applications that were listed in Section 1.

3. **Potential Practical Applications.** Let us discuss now applicability of the considered design technique. An ordinary sorting is required in many types of information processing. For a large number of data items, the known procedures that are used are time consuming and can be accelerated using the methods proposed in this paper. This is especially important for portable embedded applications. In the latter case, even sorting thousands of items can be done significantly faster in software/hardware (e.g., in APSoC) than in software only.

One common problem is clustering objects in accordance with their attributes. Different methods have been proposed for solving this problem and many of them involve sorting and searching as frequently used operations [37-40]. For example, in the CPES (Clustering with Prototype Entity Selection) method [41], a fitness function is proposed to decide if given objects can be clustered. Sorting the results produced by the fitness function enables solutions to be found faster. Actually, for many practical applications it is important to make sorting built-in, much as in operations such as computing the Hamming weights of binary vectors, e.g., POPCNT (population count) [42] and VCNT (Vector Count Set Bits) [43]. Similar proposals were made in [44].

Another example is extracting a required number of items through the Internet. For example, suppose we would like to find the N cheapest products from very large number of available options. An FPGA/APSoC based system requests and receives data from different suppliers and extracts the maximum/minimum subsets with the indicated number of items. Different search criteria can be applied and very large volumes of data can be

analyzed. Other practical applications (in which high throughput is very important) are described in [2]. One particular example can be taken from [2], which requires deciding how often the data collected falls within a set of critical values that are above or below a given threshold. Generally, the greater the intensity, the more critical is the subset and the higher the probability of an event which might happen. By discovering the maximum and minimum subsets you can determine when the activity is the highest or the lowest.

The algorithm [45] discovers rules associated with a set of classes and it has been tested on a real world application data set related to website phishing. In this algorithm the classifier sorts classes within each rule based on their frequency. Thus, sorting is also needed.

In [25] small even-odd merge and bitonic sorting networks were used to implement a median operator over a count-based sliding window. Such an operator is commonly needed to eliminate noise in sensor readings [46] and in data analysis [47], which are tasks that occur often in control engineering.

The method proposed in [48] (based on the Monte Carlo method coupled with a sorting algorithm [49] and gradient search [50]) systematically evaluates possible behaviors of a closed-loop system by analyzing its time response. This permits various techniques to be applied for solving problems that are commonly encountered in networked control systems.

The software/hardware solutions proposed in this paper are faster. They are based on two (PC - FPGA) or three (PC - APSoC: PS - PL) level systems. The effectiveness of the hardware/software solutions is underlined in [51], addressing the importance of portable computing hardware environments to handle massive data.

4. **Highly Parallel Networks for Sorting and Searching.** It is shown in Section 2.2 that sorting networks are widely used in data [25] and vector [52] processing and they enable comparison and swapping operations over multiple data items to be executed in parallel. A review of recent results in this area can be found in [26] where it is shown that many researchers and engineers consider such technique as very beneficial for data and vector processing in FPGAs and APSoCs. Although the methods [28,29] enable the fastest theoretical throughput, the actual performance is limited by interfacing circuits supplying initial data and transmitting the results and the communication overheads do not allow theoretical results to be achieved in practical designs [19].

The proposed circuits require a significantly smaller number of comparators/swappers (C/S) than networks from [28,29] and many C/S are active in parallel and reused in different iterations. The first circuit (see Figure 4) contains N M-bit registers $Rg_0, \ldots, Rg_{N-1}$. Unsorted input data are loaded to the circuit through N M-bit lines $d_0, d_1, \ldots, d_{N-1}$. For the fragment on the left-hand side of Figure 4, the number N of data items is even, but it may also be odd which is shown in the example in the same Figure 4. Each C/S is shown in Knuth notation (↕) [1] and it compares items in the upper and lower registers and transfers the item with the larger value to the upper register and the item with the smaller value to the lower register (see the upper right-hand corner of Figure 4). Such operations are applied simultaneously to all the registers linked to even C/S in one clock cycle (indicated by the letter α) and to all the registers linked to odd C/S in a subsequent clock cycle (indicated by the letter β). This implementation may be unrolled to an even-odd transition network [32], but vertical lines of C/S in Figure 4 are activated sequentially and the number of C/S is reduced compared to [32] by a factor of $N/2$. For example, if the number N is even then the circuit from [32] requires $N \times (N-1)/2$ C/S and the circuit in Figure 4 – only $N-1$ C/S. The circuit in [32] is combinational and the circuit in Figure 4 may require up to N iterations. The number N of iterations can

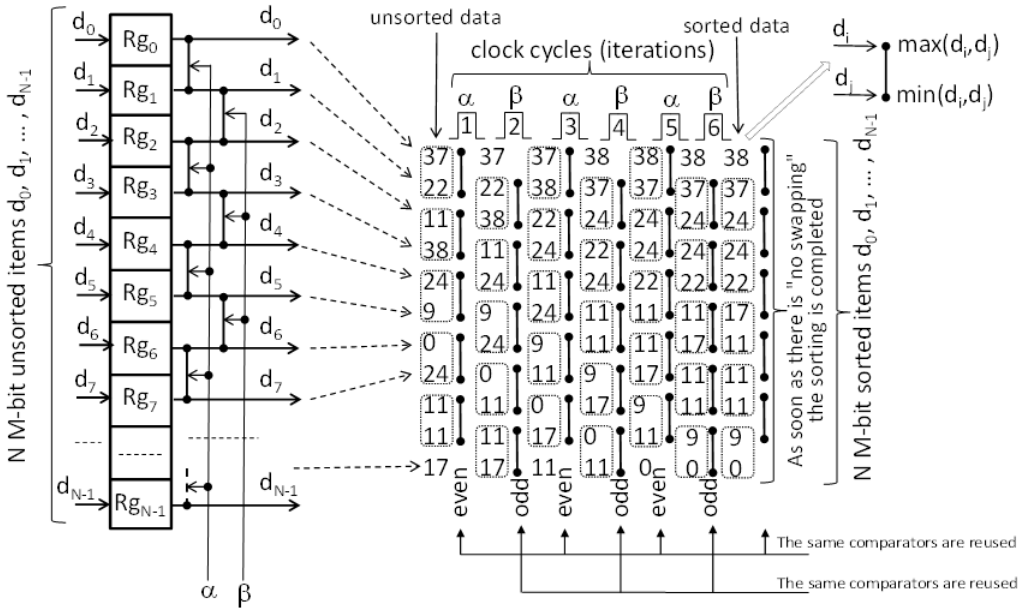

FIGURE 4. The first sorting circuit with an example

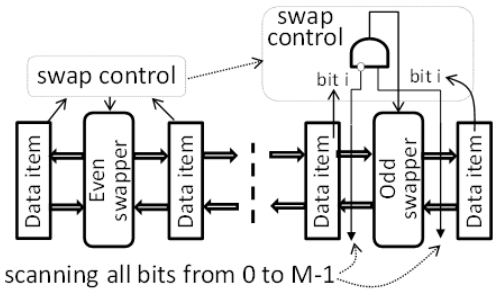

FIGURE 5. The second sorting circuit

be reduced very similarly to [26]. Indeed, if beginning from the second iteration, there is no data exchange in either even or odd C/S, then all data items are sorted. If there is no data swapping for even C/S in the first iteration, data swaps for odd C/S may still take place. Note that the network [32] possesses a long combinational delay from inputs to outputs. The circuit in Figure 4 can operate at a high clock frequency because it involves a delay of just one C/S per iteration (i.e., in each rising/falling edge of the clock).

Let us look at the example shown in Figure 4 ($N = 11$, $M = 6$). Initially, unsorted data $d_0, d_1, \ldots, d_{10}$ are copied to $Rg_0, \ldots, Rg_{10}$. Each iteration (6 iterations in total) is forced by an edge (either rising or falling) of a clock. The signal α activates the C/S between the registers $Rg_0, Rg_1, Rg_2, Rg_3, \ldots, Rg_8, Rg_9$. The signal β activates the C/S between the registers $Rg_1, Rg_2, Rg_3, Rg_4, \ldots, Rg_9, Rg_{10}$. There are 10 C/S in total. Rounded rectangles in Figure 4 indicate elements that are compared at iterations 1-6. Data are sorted in 6 clock cycles and $6 < N = 11$. Unrolled circuits from [32] would require 50 C/S with the total delay equal to the delay of N sequentially connected C/S.

The second circuit is shown in Figure 5 and it combines the first circuit with the radix sort. Now only single bits from the registers of Figure 4 are compared and if the upper bit is 0 and the lower bit is 1, then M-bit data in the relevant upper and lower registers are swapped. Thus, any C/S is built on only one gate (see Figure 5) much like in [52]. Data are sorted in such a way that at the first step bit 0 (the least significant bit) is chosen.

At the second step bit 1 is chosen and at the last step bit $M-1$ is chosen. Hence, bits in all registers are chosen sequentially, i.e., they are scanned. Clearly, the number of clock cycles is increased but the resources occupied are reduced because one-bit C/S are smaller than M-bit C/S.

The maximum number of clock cycles for sorting N M-bit items is increased up to $N \times M$ but in practical designs it is smaller because the method described above enables sorting to be completed as soon as there is no need for swapping data items.

Figure 6 demonstrates an example. The same data as in Figure 4 are chosen. Now we consider binary codes of the items. For the first sort (0) bit 0 is analyzed, which is the least significant bit. All the remaining sorts (1-5) and the relevant (analyzed) bits are shown in Figure 6. After the last step (sort 5), all data are sorted and for clarity binary codes are shown as decimal numbers.

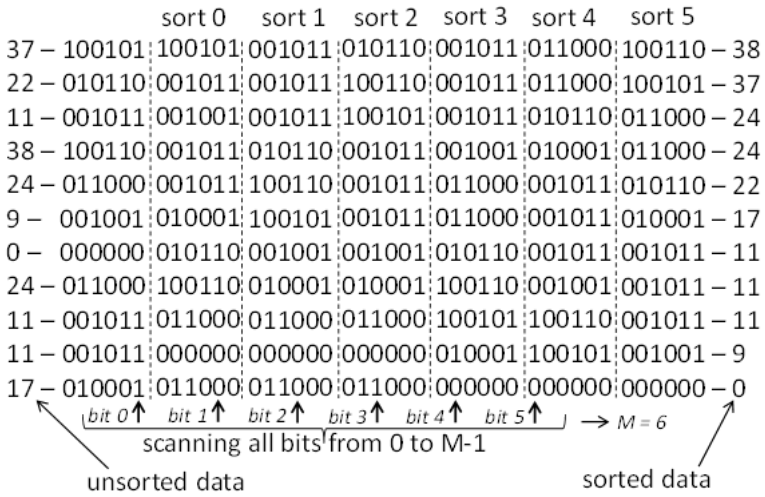

FIGURE 6. An example

The networks described above can be used efficiently for solving numerous supplementary tasks. One of these tasks is the extraction of the maximum and/or minimum subsets from the sorted sets [53-55]. Let set S containing L 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 ($L_{\max} \leq L$ and $L_{\min} \leq L$). We mainly consider such tasks for which $L_{\max} << L$ and $L_{\min} << L$, which are more common for practical applications. Since L may be very large (millions of items), the set cannot be completely processed in hardware because the resources required are not available. Figure 7 shows the top-level architecture from [55] in which the proposed iterative data sorter is used. Much like [55], the given set S is decomposed on $Q = \lceil L/K \rceil$ subsets, all of which contain exactly K M-bit items except the last, which may have less than K M-bit items. Subsets are computed incrementally in Q steps (we assume below that $K \leq L$).

At the first step, the first K M-bit data items are sorted in the network (such as that shown in Figure 4) which handles $L_{\max} + K + L_{\min}$ data items in total 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 lower parts are broken). So, sorting is done only in the middle part handling K M-bit items. As soon as 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.

From the second step, all the comparators are active, i.e., the network (see Figure 4) handles $L_{\max} + K + L_{\min}$ items. Now for each new K M-bit items, the maximum and

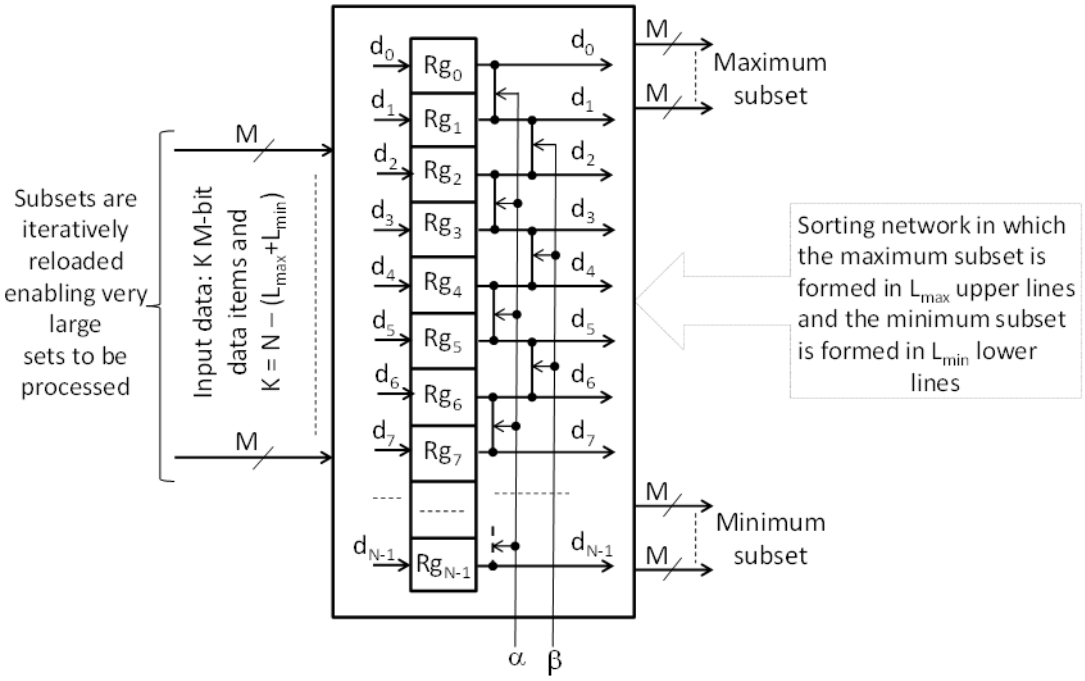

FIGURE 7. Computing the maximum and minimum sorted subsets

minimum sorted subsets are properly corrected, i.e., new items may be inserted. All other details relevant to top-level architecture in Figure 7 can be found in [55].

The next potential task that can be solved with the aid of the proposed methods is filtering [55]. 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 Figure 7 only for such data items D that fall within the bounds B_u and B_l, i.e., $B_l \le D \le B_u$ (or, possibly, $B_l < D < B_u$). Once again the basic component is the sorting network (see Figure 4) and it can be used in the method [55] that enables data items to be filtered at run-time (i.e., during data exchange).

Clearly, the operations described above can be implemented in software. For example, the C function qsort permits large data sets to be sorted. After that, extracting the maximum and minimum subsets is easily done. Filtering may be done by testing and eliminating items that do not fall within the predefined constraints. However, for many practical applications the performance of the operations described above is important. The results of thorough experiments and comparisons have shown that software/hardware solutions are significantly faster than software only solutions.

Many other problems can also be solved applying the proposed networks. For example, in [2] a highly parallel architecture was proposed that permits repeated items to be found efficiently. The basic component of the architecture [2] is a sorting network, and using the proposed technique for such a network permits the hardware resources to be reduced and the performance to be increased. Thus, the results of this paper are useful for a large number of practical applications.

5. **Software/Hardware Co-design.** Figure 8 shows the basic architecture for data transfer between a host PC and an APSoC through PCI-express.

Figure 9 presents a more detailed architecture of a three-level system for the example of distributed data sort. Software in the host PC runs the 32-bit Linux operating system (kernel 3.16) and executes programs (written in the C language) that take results from PCI-express (from the APSoC) for further processing. We assume that the data collected

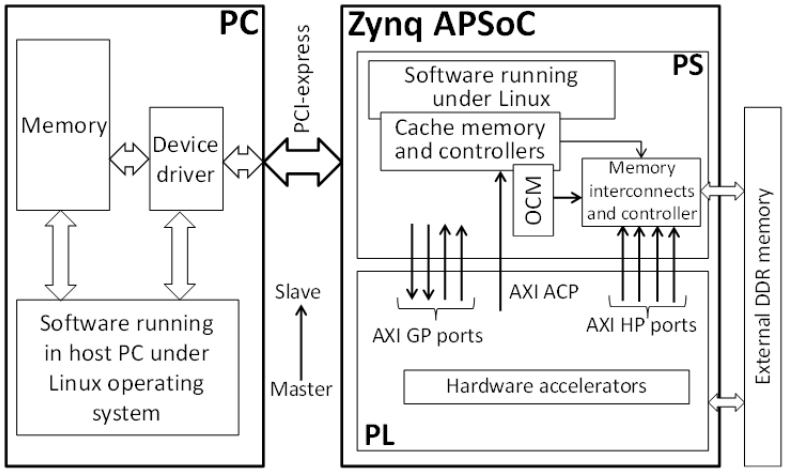

FIGURE 8. Basic architecture for data transfer between a host PC and an APSoC through PCI-express

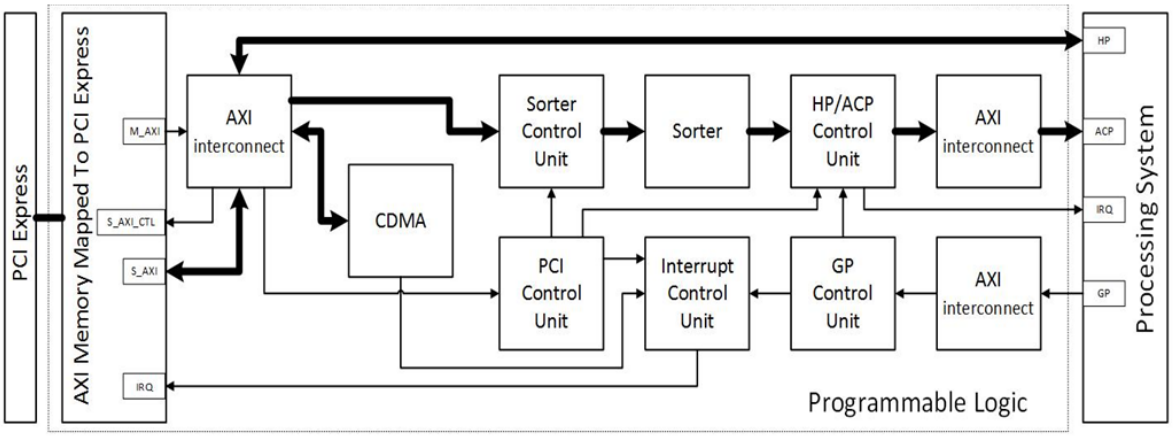

FIGURE 9. Architecture of a three-level APSoC-based system for data sorting

in the APSoC are preprocessed in the APSoC by applying various highly parallel networks (see Section 4), and the results are transferred to the host PC through the PCI-express bus. To support data exchange through PCI-express, a dedicated driver was developed. The APSoC uses the Intellectual Property (IP) core of the central direct memory access (CDMA) module [56] to copy data through AXI PCI express (AXI-PCIE) [57]. The project is similar to [58] and links CDMA and AXI-PCIE modules based on a simple data mover (i.e., the mode "scatter gather" [58] is not used). A master port (M-AXI) of the AXI-PCIE operates similarly to GP ports in [19] and supplies control instructions from the PC to customize data transfers. The instructions indicate the physical address of data for PC memory, the size of transferred data, etc. The CDMA module can be connected to either AXI HP or AXI ACP interfaces in APSoC and transmits data from either on-chip memory (OCM) or external DDR. After supplying the addresses, the number of data bytes (that need to be transferred) is indicated and the data transmission is started. As soon as the data transmission is completed, the CDMA module triggers an interrupt that has to be properly handled (the interrupt number is determined by the BIOS of the host PC). The following customization is done for 1) AXI-PCIE: legacy interrupts, 128 bits data width, and 2) CDMA: 256 bytes burst size, 128 bits data width. Note that the

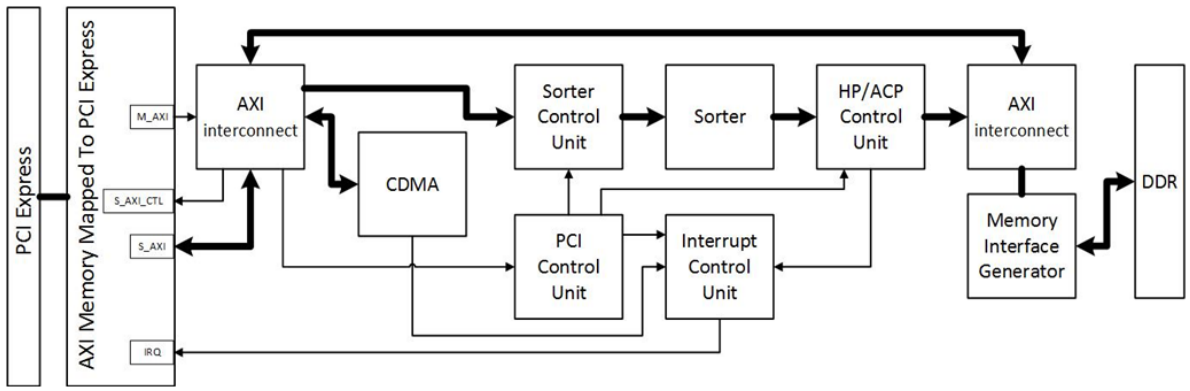

FIGURE 10. Architecture of a two-level FPGA-based system for data sorting

architecture in Figure 9 allows data transfers in both directions, i.e., data from the PC may also be received.

The architecture for the case when an FPGA is used (instead of an APSoC) is similar, and is shown in Figure 10 (now there are just two levels). The only difference is the transfer of data items from an external source to FPGA DDR memory where the data are preliminarily collected by the FPGA and stored. The DDR is controlled by a memory interface generator.

Data transfer in the host PC is organized through direct memory access (DMA). To work with different devices, a driver (kernel module) was developed. The driver creates in the directory /dev a character device file that can be accessed through read and write functions, for example write(file, data_array, data_size). Up to 5 base address registers (BAR) can be allocated but we used just one.

The PC BIOS assigns a number (an address) to the selected BAR and a corresponding interrupt number that will be later used to indicate the completion of a data transfer. As soon as the driver is loaded, a special operation (probe) is activated and the availability of the device with the given identification number (ID) is verified (the ID is chosen during the customization of the AXI-PCIE). Then a sequence of additional steps is performed (see [59, pp.302-326] for necessary details). A number of file operations are executed in addition to the probe function (see Figure 11). In our particular case, access to the file is done through read/write operations. Figure 11 demonstrates the interaction of a user application with the driver (kernel module) and some additional operations that may be executed.

As soon as a user program calls the read function, the read(file, data_array, data_size) function gets the address in the user memory space and the number of bytes that need to be transferred. Initially, the data are copied to a buffer and then the physical address of the buffer is obtained. Now the data are ready to be transferred from APSoC/FPGA. Then the data are copied and the driver is waiting for an interrupt indicating that the data transmission is complete. The necessary operations for generating the interrupt are given in [56]. Additional details can be found in [59, pp.258-287].

The proposed networks (see Section 4) can be used as follows. The sorter receives blocks composed of N M-bit data items that are collected from sensors initially and stored in memories (such as external DDR and OCM). Interactions with memory are done through AXI HP/ACP ports (see Figure 9) or through the memory interface block (see Figure 10). The sorter (such as that shown in Figure 4) executes iterative operations over multiple parallel data and is controlled by a dedicated finite state machine (FSM) called Sorter Control Unit (see Figures 9 and 10). The ports are also controlled by a dedicated FSM

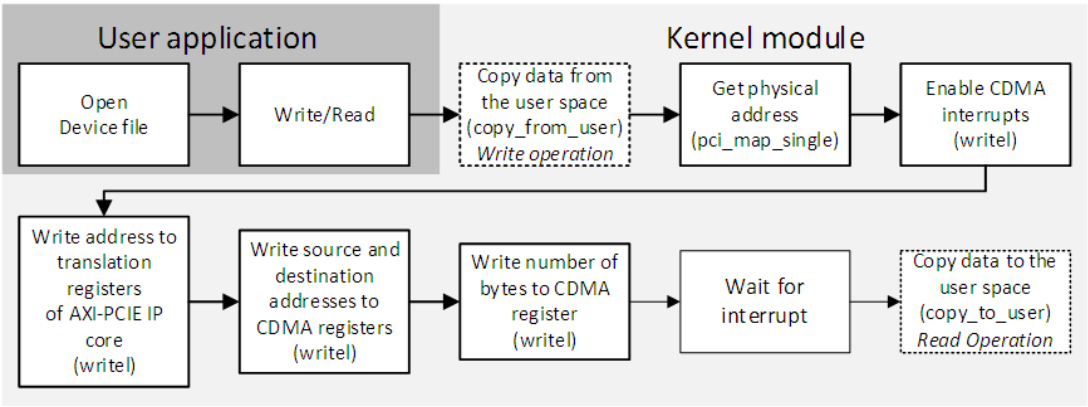

FIGURE 11. Operations with the device driver in the host PC and additional operations that may be executed

(see HP/ACP Control Unit in Figures 9 and 10). The results of sorting are copied back to memory and then transmitted to the host PC through the PCI-express bus. APSoC PS is responsible for data collection and organization that is done in accordance with the established requirements. For the case of FPGA, data collection and organization are done by specially developed dedicated circuits. Finally, either the PS or the dedicated circuits prepare data in memory so that these data can be processed in the PL/FPGA and the results of the processing (stored in memory) are ready to be transmitted to the host PC. The blocks CDMA with control units (PCI Control Unit and Interrupt Control Unit in Figures 9 and 10) are responsible for transmitting data.

6. **Experiments and Comparisons.** The system for data transfers between a host PC and an APSoC/FPGA has been designed, implemented, and tested. Experiments were done with two prototyping boards. The first is the Xilinx ZC706 evaluation board [7] containing the Zynq-7000 XC7Z045 APSoC device with PCI express endpoint connectivity "Gen1 4-lane (x4)". The PS is the dual-core ARM Cortex-A9 and the PL is a Kintex-7 FPGA from the Xilinx 7th series. The second board is VC707 [8] and it contains the Virtex-7 XC7VX485T FPGA from the Xilinx 7th series with PCI express endpoint connectivity "Gen2 8-lane (x8)". All designs were done for: 1) hardware in the PL of APSoC/FPGA synthesized from specifications in VHDL that describe circuits interacting with Xilinx IP cores (Xilinx Vivado Design Suite 2015.1/2015.2); 2) software in the PS of APSoC developed in C language (Xilinx Software Development Kit – SDK 2015.1); 3) user programs running under the Linux operating system in the host PC developed in C. Data were transferred from the ZC706/VC707 to the host PC through PCI-express. The host PC contains Intel core i7 3820 3.60GHz.

Figure 12 demonstrates organization of experiments with data sorters.

We assume that data are collected by the ZC706/VC707 board and stored in DDR memory (in the experiments, data are produced as described in point 1 below). Subsequently, different components (A, B, C, D) may be involved in data processing:

1) Data are randomly generated in the programmable logic (in the PL of APSoC or in the FPGA) and sorted using only networks in hardware (component A), indicated below as *Sorting blocks*;
2) Data are transferred from the ZC706/VC707 to the PC through PCI-express and sorted by software in the PC (component D), indicated below as *PC sort*;
3) Data are completely sorted in the APSoC (the set of data items is decomposed into blocks, blocks are sorted in the PL by the networks described above, the sorted blocks

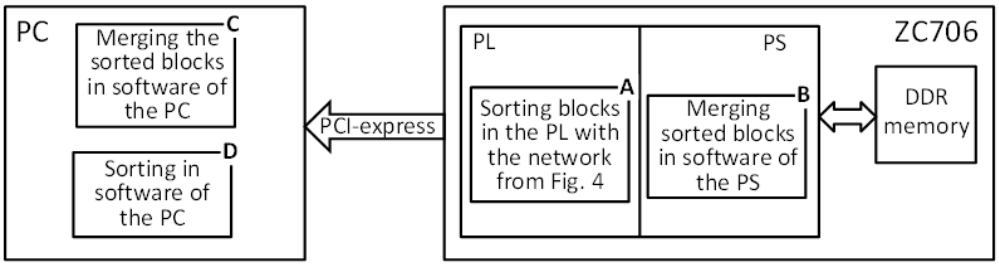

FIGURE 12. Organization of experiments with data sorters (the size of one block is 1024 32-bit data items)

are merged in the PS to produce the final result) and the sorted data are transferred to the PC through PCI-express (components A and B), indicated below as *Sorting + PS merge*;

4) Data are completely sorted in APSoC/FPGA and in the PC in such a way that: a) blocks of data are sorted in the PL of APSoC or in FPGA; b) the sorted blocks are transferred to the PC through PCI-express; and c) the blocks are merged by software in the PC (components A and C). This case is indicated below as *Sorting + PC merge*.

Sorting in hardware only (see point 1 above) permits the circuits that process the maximum possible number of data items and can be entirely implemented in the programmable logic without any support from software to be evaluated. In the next subsections we will present the following results: 6.1) evaluation of the circuits including threshold values that are potential limitations of the methods proposed; 6.2) comparisons with the best known alternatives.

6.1. **Evaluation of the proposed circuits.** Evaluation of the proposed circuits has been done through a set of experiments with the network from Figure 4, selecting four data sets sizes of 512, 1024, 2048, and 4096 items. The results are shown in Figure 13.

We counted only the percentage of look-up tables (LUTs), which are the primary PL/FPGA resources that are used for the network. The percentage of other resources is lower, for example, the percentage of flip-flops for the FPGA does not exceed 23% and

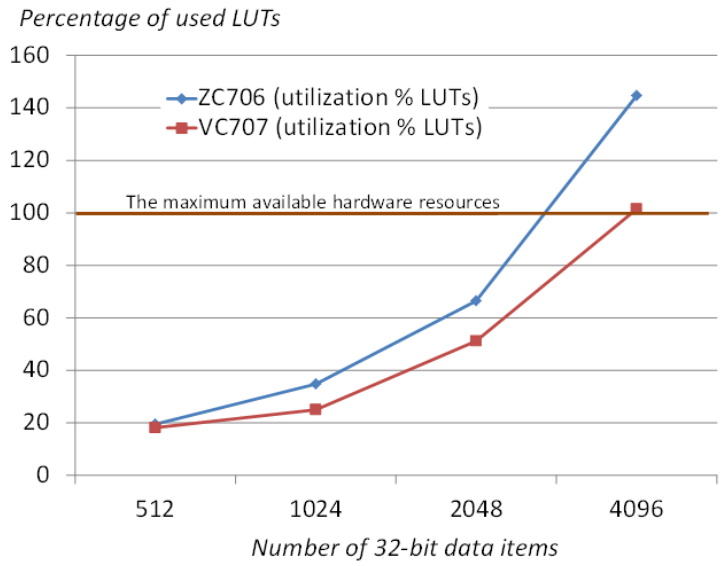

FIGURE 13. The results of sorting in hardware only using iterative networks from Figure 4

for the PL – 31% for all data set sizes (from 512 to 4096). From Figure 13 we can see that the available resources permit only iterative networks of up to 2048 32-bit data items to be implemented. Thus 2048 is the threshold for hardware only implementations based on the microchips indicated above. A preliminary evaluation shows that 8192 items is the maximum threshold value for hardware-only implementations of the circuit from Figure 4 in the most advanced FPGAs/APSoCs currently available on the market.

The circuit in Figure 5 was also implemented and tested. As we expected in Section 4, the occupied resources were reduced, and the time was increased compared to the network in Figure 4 by a factor of about M. For example, if the size of one block is 1024 ($N = 1024$) of 32-bit ($M = 32$) items, then the percentage of LUTs used for VC707 is 25% for the network in Figure 4, and 18% for the network in Figure 5. Thus, sorting 4096 items in the FPGA of VC707 is possible. However, we found that the remaining resources are not sufficient to implement the other blocks shown in Figure 14. Thus, the circuit in Figure 5 makes sense only for autonomous hardware networks.

In further experiments (see Figure 12) only the network from Figure 4 will be used. The size of data varies from 2 KB to 1024 KB ($M = 32$). The results obtained for the four measurements indicated above are reported in Figure 14 (the two curves *PC sort* and *PC sort + data transfer* show the same results without and with data transfers). The result for each type of experiment is an average of 64 runs.

The following conclusions can be drawn from Figure 14.

- The fastest results were obtained for the components A and C, i.e., pre-sort in the PL with a subsequent merge in the PC (see point 4 above). Note that the fastest (the lowest) curve in Figure 14 is built for sorting individual subsets only. Thus, the complete data set has not been sorted and the relevant results cannot be used for comparisons.
- The slowest result is shared between the remaining two cases (see points 2, 3 above).
- Note that for almost all data sizes, sorting and merging in APSoC is faster than sorting in PC software. Thus, cheaper (than PC) APSoCs are more advantageous and may be used efficiently for embedded applications.
- Sorting blocks in the PL network (see Figure 4) is significantly faster than subsequent merging. All communication and protocol overheads were taken into account.

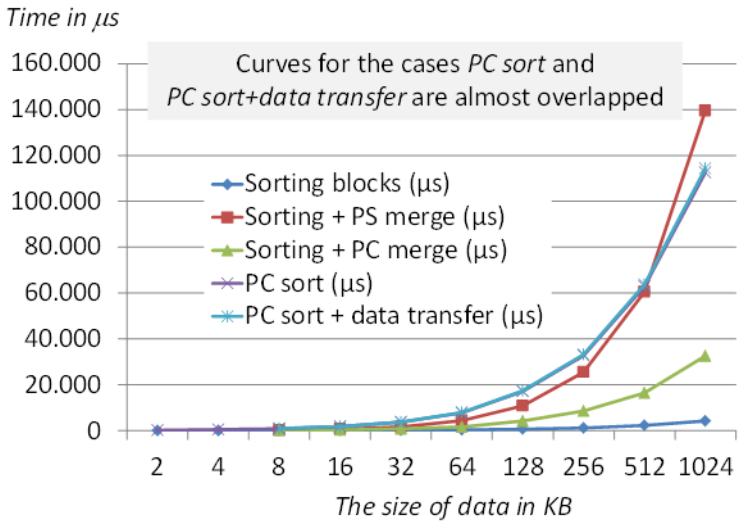

FIGURE 14. The results of experiments with the three-level system sorting data (the size of one block is 1024 32-bit data items)

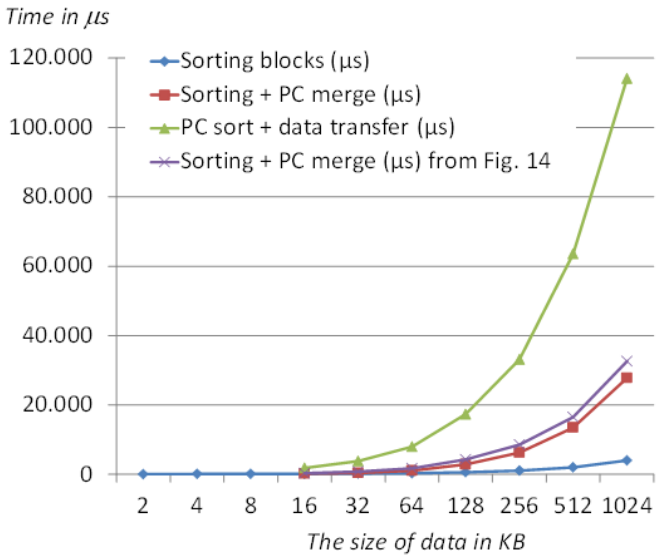

FIGURE 15. The results of experiments with the two-level system sorting data (the size of one block is 2048 32-bit data items)

Similar experiments were done with the VC707 prototyping board, but with the blocks of data containing 2048 32-bit data items (i.e., the blocks sorted in the hardware network are two times larger). The results are shown in Figure 15.

From analyzing these results we can conclude that:

- Using an FPGA from the Virtex-7 family, sorting in hardware networks is slightly faster, but the difference is negligible;
- Using larger blocks (2048 vs. 1024) allows sorting in point 4 (see the beginning of this section) to be faster by a factor ranging from 1.2 to 1.8. This is because the depth of software merges is reduced by one level.

The next experiments were done extracting the maximum and the minimum sorted subsets. We found that the acceleration is better than in Figures 14 and 15 for data sorting. This is because the number of data transferred through PCI express is significantly decreased and almost all operations are done in the APSoC/FPGA. We implemented and tested the circuit shown in Figure 7 in the PL of APSoC, which takes data from the DDR memory and extracts the maximum and minimum subsets with $L_{\max}/L_{\min}$ data items, where $L_{\max}/L_{\min}$ varies from 128 to 1024 (as before $M = 32$, L varies from 2 KB to 1024 KB). Table 1 presents the results for $L_{\max}/L_{\min} = 128$.

TABLE 1. The results of experiments extracting the maximum/minimum subsets

Data (KB)	Time (μs)	Data (KB)	Time (μs)
2	70	64	254
4	75	128	425
8	89	256	916
16	112	512	1543
32	157	1024	3535

For some practical applications the maximum and/or minimum subsets may be large and the available hardware resources become insufficient to implement the circuit from Figure 7 [54]. The problem can be solved by applying the following technique. Let $l_{\max}$ and $l_{\min}$ be constraints for the upper and lower parts of the sorting network in Figure 7,

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 this case, the maximum/minimum subsets can be computed incrementally as follows [54].

1. In 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. This correction avoids the loss of repeated items in subsequent steps. Indeed, the minimum value from the maximum subset (the maximum value from the minimum subset) can appear in subsets that are generated in point 3 below, and they will be lost because of filtering (see point 3 below).
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 a general-purpose port.
3. The same data items (from memory), as in point 1 above, are initially filtered [55] so 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 the subsets from point 1).

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.

If the number of repeated items is greater than or equal to $l_{\max}/l_{\min}$, then the method above may generate infinite loops. This situation can easily be recognized. Indeed, if any new subset becomes empty after the corrections in point 1 above, then an infinite loop will be created. In this case, we can use the previously described method based on software/hardware sorters (i.e., sorting in hardware and subsequent merging in software). Thus, data items are sorted before the desired number of the largest and/or the smallest items are taken.

Table 2 presents the results for larger numbers of data items in extracted subsets (from 128 to 1024) for $L = 256$ KB.

TABLE 2. The results of experiments with extracting subsets with different number of data items

Data	Time (μs)	Data	Time (μs)
$128 + 128$	916	$640 + 640$	4481
$256 + 256$	1808	$768 + 768$	5372
$384 + 384$	2698	$896 + 896$	6261
$512 + 512$	3589	$1024 + 1024$	7152

The developed software and hardware can also solve higher level tasks. As examples, we considered creating objects in software for further clustering and finding the frequency of occurrence of data items in hardware. The attributes of any individual object are generated randomly in software within a given range. Objects and attributes are associated with rows and columns of a matrix. Clearly, the Hamming weight of any row r indicates how many times the attribute associated with r appeared in different objects (associated with columns). Two tasks are solved in the PL: 1) calculating the Hamming weights using the methods and tools from [2]; and 2) sorting the Hamming weights with the aid of the methods described above. The sorted values are used to simplify solving different problems from the scope of data mining.

Finding the item that occurs most frequently can be done entirely in hardware. Suppose we have a set of L sorted data items which may include repeated items and we need the most frequently repeated item to be found. This problem is solved in the hardware circuit proposed in [2]. Thus, combining the proposed solutions with the circuit [2] enables the complete problem to be solved.

6.2. **Comparisons with the best known alternatives.** Comparisons with the best known alternatives can be done by analyzing the fastest known networks. For data sorting, the latency and the cost of the most widely discussed networks are shown in Table 3. The formulae for the table are taken from [1,23,25,26,32]. For example, if $N = 1024$ then the latency is equal to $D(1024) = 55$ for the fastest known even-odd merge and bitonic merge networks [28,29], which is smaller than the number of iterations for the proposed network. However, $C(1024)$ for the less resource consuming even-odd merge network is 24,063 C/S and for the proposed network $C(1024) = 1023$ C/S. Thus, the difference is a factor of about 24. It means that with the same hardware resources, the proposed networks can process blocks of data with significantly larger number N of data items. Indeed, the resources $C(1024) = 24,063$ of the known even-odd merge network are the same as for 24 proposed networks each of which sorts the same number of data items, i.e., 1024. This means that the proposed network occupies less than 5% of the resources of the known network and the number of sorted items is exactly the same.

TABLE 3. Cost $C(N)$ and latency $D(N)$ of the most widely discussed networks

Type of the network	$C(N)$	$D(N)$
Bubble and insertion sort	$N \times (N-1)/2$	$2 \times N - 3$
Even-odd transition	$N \times (N-1)/2$	N
Even-odd merge	$\left(p^2 - p + 4\right) \times 2^{p-2} - 1$, $N = 2^p$	$p \times (p+1)/2$, $N = 2^p$
Bitonic merge	$\left(p^2 + p\right) \times 2^{p-2}$, $N = 2^p$	$p \times (p+1)/2$, $N = 2^p$
The proposed network (see Figure 4)	$N - 1$	$\leq N$

The experiments done for the board [8] have shown that for the networks [28,29] $N \leq 128$, while for the proposed networks $N > 2048$. Thus, the proposed networks may handle about 16 times larger blocks. The blocks created in hardware are further merged in software, thus the number of levels in software will be increased in the known networks by a factor of $\lceil \log_2 16 \rceil = 4$ (comparing to the proposed network). The following experiments were done:

1. Blocks with two sizes (that are 128 and 2048 32-bit words) have been sorted in software using the known (for the size 128) and the proposed (for the size 2048) networks. The measured times are T_{128} and T_{2048}.
2. Since the known networks cannot be used for $N = 2048$, the same results have been obtained through a subsequent merge in software of blocks with $N = 128$ to get blocks with $N = 2048$. The measured time is $T_{128} + T_{\text{merge}}$.
3. Finally we measured the value $(T_{128} + T_{\text{merge}})/T_{2048}$. The fastest method was used i.e., pre-sort in the PL with subsequent merge in the PC (see Subsection 6.1). The result that was an average of 64 runs exceeds 5. Note that additional delays appeared also in data transmission through PCI-express of smaller blocks of data items.

For subsequent merging required for larger data sets all the conditions for the proposed and known methods are the same. Thus, the proposed methods are always faster because merging in software begins with significantly larger pre-sorted blocks. Clearly, threshold

values for maximum sizes of sorted sets are the same as for general-purpose software running in a PC.

Comparison of the proposed methods for extracting the maximum and minimum sorted subsets with the results in [53] demonstrates that the proposed method permits significantly larger subsets to be constructed. Indeed, the maximum size of extracted subsets in [53] is only 8 items and the maximum size of initial set is only 256 items. The size of each item is 10 bits. This is because the methods [53] are based on even-odd merge and bitonic merge networks for which the complexity of the circuits, i.e., the value of $C(N)$, is limited. In our case, the maximum size of extracted subsets is 1024 (which exceeds the size of initial data sets in [53]) and the size of initial set is up to 1024 KB. The size of each item is 32 bits (versus 10 in [53]). The conclusion is the following: 1) the proposed methods enable data sets with significantly larger numbers of items to be processed; 2) the size of the extracted (minimum, maximum, or both) subsets may be increased in the proposed networks; 3) the performance (throughput) for processing large subsets in the proposed methods is better because complex tasks cannot be entirely solved in hardware using the methods [53] and the necessary software introduces large additional delays.

7. **Conclusions.** The paper is dedicated to distributed computing systems that involve higher level computers (such as a PC) interacting with programmable systems-on-chip through a PCI-express bus. We studied different levels of such systems, namely higher level software running in the host PC, data transfer through PCI-express, lower level software running in ARM of a programmable system-on-chip, and hardware accelerators implemented in the programmable logic. It is shown that sorting and searching are common operations in different types of data and information processing. We found the fastest way to implement such operations and suggested numerous supplementary operations that are common to different computing systems. A number of hardware accelerators were proposed, all of which were completely implemented and tested in commercial computers and microelectronic devices. The practical analysis consisted of numerous experiments using the most recent all programmable systems-on-chip combining a processing system with configurable logic. The experiments comprehensively demonstrated that the proposed multilevel solutions outperformed software running in a PC by a significant margin. It is also shown that the networks proposed can be used in numerous practical applications in control engineering and applied informatics.

Acknowledgments. The authors would like to thank Ivor Horton for his very useful comments. This research was supported by the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research, the Study IT in Estonia Programme, and Portuguese National Funds through FCT – Foundation for Science and Technology, in the context of the projects UID/CEC/00127/2013 and Incentivo/EEI/UI0127/2014.

REFERENCES

[1] D. E. Knuth, *The Art of Computer Programming, Sorting and Searching, Vol. III*, Addison-Wesley, 2011.

[2] V. Sklyarov and I. Skliarova, Digital hamming weight and distance analyzers for binary vectors and matrices, *International Journal of Innovative Computing, Information and Control*, vol.9, no.12, pp.4825-4849, 2013.

[3] D. Zmaranda, H. Silaghi, G. Gabor and C. Vancea, Issues on applying knowledge-based techniques in real-time control systems, *International Journal of Computers, Communications and Control*, vol.8, no.1, pp.166-175, 2013.

[4] L. Field, T. Barnie, J. Blundy, R. A. Brooker, D. Keir, E. Lewi and K. Saunders, Integrated field, satellite and petrological observations of the November 2010 eruption of Erta Ale, *Bulletin of Volcanology*, vol.74, no.10, pp.2251-2271, 2012.

[5] W. Zhang, K. Thurow and R. Stoll, A knowledge-based telemonitoring platform for application in remote healthcare, *International Journal of Computers, Communications and Control*, vol.9, no.5, pp.644-654, 2014.
[6] Z. K. Baker and V. K. Prasanna, An architecture for efficient hardware data mining using reconfigurable computing systems, *Proc. of the 14th Annual IEEE Symp. on Field-Programmable Custom Computing Machines*, pp.67-75, 2006.
[7] Xilinx, Inc., *ZC706, All Programmable SoC Evaluation Kit (Vivado Design Suite 2014.3)*, http://www.xilinx.com/support/documentation/boards_and_kits/zc706/2014_3/ug961-zc706-GSG.pdf.
[8] Xilinx, Inc., *VC707 Evaluation Board for the Virtex-7 FPGA User Guide*, http://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf.
[9] R. Rajkumar, I. Lee, L. Sha and J. Stankovic, Cyber-physical systems: The next computing revolution, *Proc. of the 47th ACM/IEEE Design Automation Conference*, pp.731-736, 2010.
[10] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems – A Cyber-Physical Systems Approach*, 2nd Edition, 2011.
[11] J. C. Jensen, E. A. Lee and S. A. Seshia, *An Introductory Lab in Embedded and Cyber-Physical Systems*, http://leeseshia.org/lab, 2014.
[12] K. Vipin, S. Shreejith, S. A. Fahmy and A. Easwaran, Mapping time-critical safety-critical cyber physical systems to hybrid FPGAs, *Proc. of the 2nd IEEE International Conference on Cyber-Physical Systems, Networks, and Applications*, pp.31-36, 2014.
[13] M. Panunzio and T. Vardanega, An architectural approach with separation of concerns to address extra-functional requirements in the development of embedded real-time software systems, *Journal of Systems Architecture*, vol.60, pp.770-781, 2014.
[14] A. Borangiu and D. Popescu, Digital signal processing for knowledge based sonotubometry of eustachian tube function, *Journal of Control Engineering and Applied Informatics*, vol.16, no.3, pp.56-64, 2014.
[15] Y. Benmoussa, J. Boukhobza, E. Senn, Y. Hadjadj-Aoul and D. Benazzouz, A methodology for performance/energy consumption characterization and modeling of video decoding on heterogeneous SoC and its applications, *Journal of Systems Architecture*, vol.61, pp.49-70, 2015.
[16] M. Santarini, Products, profits proliferate on ZynqSoC platforms, *XCell Journal*, vol.88, pp.8-15, 2014.
[17] M. Santarini, Xilinx 16nm UltraScale+ devices yield 2-5X performance/watt advantage, *XCell Journal*, vol.90, pp.8-15, 2015.
[18] Xilinx Inc., *Zynq-7000 All Programmable SoC Technical Reference Manual*, http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.
[19] J. Silva, V. Sklyarov and I. Skliarova, Comparison of on-chip communications in Zynq-7000 all programmable systems-on-chip, *IEEE Embedded Systems Letters*, vol.7, no.1, pp.31-34, 2015.
[20] R. Budruk, D. Anderson and T. Shanley, *PCI-Express System Architecture*, Addison-Wesley, 2008.
[21] N. Edwards, *Theoretical vs. Actual Bandwidth: PCI Express and Thunderbolt*, http://www.tested.com/tech/457440-theoretical-vs-actual-bandwidth-pci-express-and-thunderbolt/, 2013.
[22] J. Lawley, *Understanding Performance of PCI Express Systems*, http://www.xilinx.com/support/documentation/white_papers/wp350.pdf, 2014.
[23] R. Mueller, J. Teubner and G. Alonso, Sorting networks on FPGAs, *The International Journal on Very Large Data Bases*, vol.21, no.1, pp.1-23, 2012.
[24] J. Ortiz and D. Andrews, A configurable high-throughput linear sorter system, *Proc. of IEEE Int. Symp. on Parallel & Distributed Processing*, pp.1-8, 2010.
[25] R. Mueller, *Data Stream Processing on Embedded Devices*, Ph.D. Thesis, ETH, Zurich, 2010.
[26] V. Sklyarov and 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.
[27] M. Zuluada, P. Milder and M. Puschel, Computer generation of streaming sorting networks, *Proc. of the 49th Design Automation Conference*, pp.1245-1253, 2012.
[28] K. E. Batcher, Sorting networks and their applications, *Proc. of AFIPS Spring Joint Computer Conference*, pp.307-314, 1968.
[29] S. W. Aj-Haj Baddar and K. E. Batcher, *Designing Sorting Networks: A New Paradigm*, Springer, 2011.
[30] S. Lacey and R. Box, A fast, easy sort: A novel enhancement makes a bubble sort into one of the fastest sorting routines, *Byte*, vol.16, no.4, pp.315-320, 1991.

[31] R. D. Chamberlain and N. Ganesan, Sorting on architecturally diverse computer systems, *Proc. of the 3rd International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, pp.39-46, 2009.

[32] P. Kipfer and R. Westermann, *GPU Gems*, Improved GPU Sorting, http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter46.html.

[33] C. Grozea, Z. Bankovic and P. Laskov, FPGA vs. multi-core CPUs vs. GPUs, in *Facing the Multicore-Challenge*, R. Keller, D. Kramer and J. P. Weiss (eds.), Springer-Verlag, 2010.

[34] B. Cope, P. Y. K. Cheung, W. Luk and L. Howes, Performance comparison of graphics processors to reconfigurable logic: A case study, *IEEE Trans. Computers*, vol.59, no.4, pp.433-448, 2010.

[35] V. Sklyarov, I. Skliarova, J. Silva, A. Rjabov, A. Sudnitson and C. Cardoso, *Hardware/Software Co-design for Programmable Systems-on-Chip*, TUT Press, 2014.

[36] V. Sklyarov, I. Skliarova, J. Silva and A. Sudnitson, Design space exploration in multi-level computing systems, *Proc. of the 15th International Conference on Computer Systems and Technologies*, pp.40-47, 2014.

[37] S. Sun, *Analysis and Acceleration of Data Mining Algorithms on High Performance Reconfigurable Computing Platforms*, Ph.D. Thesis, Iowa State University, 2011.

[38] S. Sun and J. Zambreno, Design and analysis of a reconfigurable platform for frequent pattern mining, *IEEE Trans. Parallel and Distributed Systems*, vol.22, no.9, pp.1497-1505, 2011.

[39] X. Wu, V. Kumar, J. R. Quinlan et al., Top 10 algorithms in data mining, *Knowledge and Information Systems*, vol.14, no.1, pp.1-37, 2014.

[40] M. F. Firdhous, Automating legal research through data mining, *International Journal of Advanced Computer Science and Applications*, vol.1, no.6, pp.9-16, 2010.

[41] E. Kovacs and I. Ignat, Clustering with prototype entity selection compared with K-means, *Journal of Control Engineering and Applied Informatics*, vol.9, no.1, pp.11-18, 2007.

[42] Intel, Corp., *Intel® SSE4 Programming Reference*, http://home.ustc.edu.cn/∼shengjie/REFERENCE/sse4_instruction_set.pdf, 2007.

[43] ARM, Ltd., *NEON™ Version: 1.0 Programmer's Guide*, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0018a/index.html, 2013.

[44] O. Arnold, S. Haas, G. Fettweis, B. Schlegel, T. Kissinger and W. Lehner, An application-specific instruction set for accelerating set-oriented database primitives, *Proc. of ACM SIGMOD International Conference on Management of Data*, pp.767-778, 2014.

[45] N. Abdelhamid, Multi-label rules for phishing classification, *Applied Computing and Informatics*, vol.11, pp.29-46, 2015.

[46] L. R. Rabiner, R. Marvin, M. R. Sambur and C. E. Schmidt, Applications of a nonlinear smoothing algorithm to speech processing, *IEEE Trans. Acoustics, Speech and Signal Processing*, vol.23, no.6, pp.552-557, 1975.

[47] J. W. Tukey, *Exploratory Data Analysis*, Addison-Wesley, 1977.

[48] A. P. Batista and F. G. Jota, Effects of time delay statistical parameters on the most likely regions of stability in an NCS, *Journal of Control Engineering and Applied Informatics*, vol.16, no.1, pp.3-11, 2014.

[49] R. Sedgewick, Implementing quicksort programs, *Communications of the ACM*, vol.21, no.10, pp.847-857, 1978.

[50] Y. Yuan, Step-sizes for the gradient method, *Proc. of the 3rd International Congress of Chinese Mathematicians*, pp.785-796, 2008.

[51] A. Goodman, Perspective emerging topics and challenges for statistical analysis and data mining, *Statistical Analysis and Data Mining*, vol.4, pp.3-8, 2011.

[52] S. J. Piestrak, Efficient hamming weight comparators of binary vectors, *Electronic Letters*, vol.43, no.11, pp.611-612, 2007.

[53] A. Farmahini-Farahani, H. J. Duwe, M. J. Schulte and K. Compton, Modular design of high-throughput, low-latency sorting units, *IEEE Trans. Computers*, vol.62, no.7, pp.1389-1401, 2013.

[54] V. Sklyarov, I. Skliarova, A. Rjabov and A. Sudnitson, Zynq-based system for extracting sorted subsets from large data sets, *Journal of Microelectronics, Electronic Components and Materials*, vol.45, no.2, pp.142-152, 2015.

[55] V. Sklyarov, I. Skliarova, A. Rjabov and A. Sudnitson, Computing sorted subsets for data processing in communicating software/hardware control systems, *International Journal of Computers Communications & Control*, vol.11, no.1, pp.126-141, 2016.

[56] Xilinx, Inc., *AXI Central Direct Memory Access v4.1*, http://www.xilinx.com/support/documentation/ip_documentation/axi_cdma/v4_1/pg034-axi-cdma.pdf, 2015.

[57] Xilinx, Inc., *LogiCORE IP AXI Bridge for PCI Express v1.06*, http://www.xilinx.com/support/documentation/ip_documentation/axi_pcie/v2_5/pg055-axi-bridgepcie.pdf, 2012.
[58] Xilinx, Inc., *PCI Express Endpoint-DMA Initiator Subsystem*, http://www.xilinx.com/support/documentation/application_notes/xapp1171-pcie-central-dma-subsystem.pdf, 2013.
[59] J. Corbet, A. Rubini and G. Kroah-Hartman, *Linux Device Drivers*, http://lwn.net/Kernel/LDD3/.