ISSN 0005-1179, Automation and Remote Control, 2017, Vol. 78, No. 1, pp. 100–112. © Pleiades Publishing, Ltd., 2017.
Original Russian Text © V. Sklyarov, I. Skliarova, 2017, published in Avtomatika i Telemekhanika, 2017, No. 1, pp. 121–136.

LOGICAL CONTROL

Data Processing in the Firmware Systems for Logic Control Based on Search Networks

V. Sklyarov[a] and I. Skliarova[b]

Universidade Aveiro, Instituto di Electrónica et Informática, Aveiro, Portugal
e-mail: [a] skl@ua.pt, [b] iouliia@ua.pt

Received October 12, 2015

Abstract—It was proposed to use the hardware accelerators for analysis and data processing in the systems of logic control on a chip including the interacting processor system, memory, and configurable logic components. The data processing expected execution of operations over the sets of elements each of which can be activated by software and realized in the hardware in parallel networks admitting, if necessary, pipeline processing. New methods of design and use of the sorting and search networks were proposed, and the results of their theoretical and experimental comparison with the existing networks were presented.

Key words: data processing, search networks, hardware accelerators, firmware systems.

DOI: 10.1134/S0005117917010088

1. INTRODUCTION

Models of the controlling and controlled devices find wide use in the problems of control. The control devices are more and more realized in the firmware systems enabling one to merge optimally the high hardware speed and software flexibility. The controlled device executes the instructions of the controlling one and provides data to select and generate the necessary instructions. Such approach is widely used in the so-called cyber-physical systems (CPS). This term was recently suggested to underline the interactive integration of calculations and physical processes [1]. Many methods of CPS design make use of the block diagrams of algorithms and discrete devices such as the finite automata [1, 2]. CPS underlie many medical devices, systems to control the transport facilities and highways, navigation equipment, robots, control systems for industrial processes, environmental analyzers, and so on [3, 4]. For example, [4] describes adaptive systems for *adaptive cruise control* and *collision avoidance* which get periodic signals from sensors such as the radars, lidars (Light Identification Detection and Ranging), or video cams, and process data to extract (filter out) the desired information. Analysis of this information (data) in the control algorithm allows one to determine the force of acceleration, deceleration, and the trajectory of the transportation facility. This functionality is time-critical, and the programs executed in the computing device are not always capable to meet the desired criteria for speed. Independently of the application area, the typical CPS system processes the data from the sensors, takes decision, and executes the control functions by sending instructions to the actuators and mechanisms. In the software and hardware systems analysis and data processing can be organized in different ways [5]. For these purposes, frequently used are the priority buffers for selection of the desired segments of the algorithms [1], extraction of data corresponding to certain criteria [6], filtering out the sensor data [7], image processing [8], accumulation and ordering of the data from the technological equipment [9], statistical analysis, and so on. For example, it is required to determine values which occur together maximal or minimal number of times over a certain time interval.

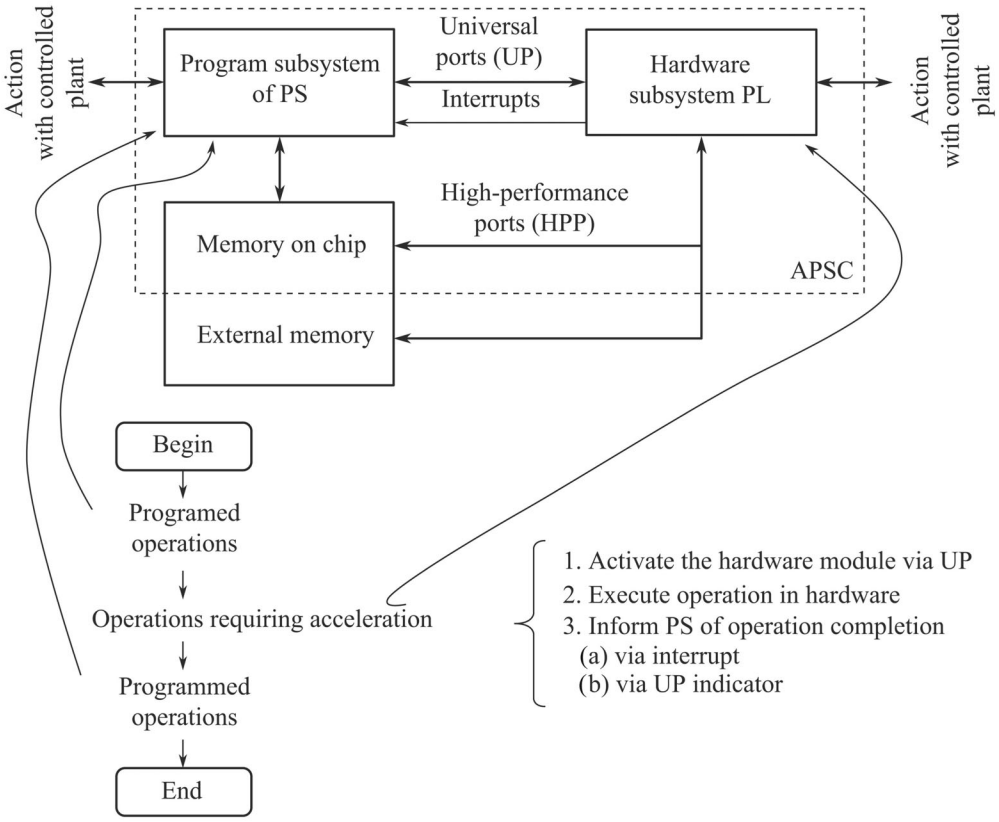

Fig. 1. Structure of the firmware system (a); execution of operations requiring acceleration (b).

The control functions can be described in different ways; for example, in terms of the unified modeling language (UML), where the designer describes the desired behavior of the system and generates the program code [1, 5], and in terms of the rearrangeable hierarchical automata (HA) [10, 11] that can perform various control functions [12]. Such methods were used earlier [13]. The HA model can be realized in software and hardware, and there exist methods to convert the corresponding programs to the hardware and vice versa [14] which allows one to satisfy various requirements on speed.

The CPS's can be efficiently realized on the all programmable system-on-chip (APSC) [4] such as Zynq-7000 proposed by Xilinx in 2011 [15] and improved appreciably in 2015 [16]. The basic module Zynq-7000 has a two-kernel processor system (PS) based on the advanced RISC Machine (ARM) Cortex-A9 and programmable logic (PL) from the seventh series of Xilinx devices or, more specifically, field programmable gate array (FPGA) of the family Artix-7 or Kintex-7. PS and PL are connected by nine high-performance interfaces with theoretical data transmission speed up to 1200 Mb per second for the write and read modes which can be executed concurrently [17]. As was shown in [18], the actual data transmission speed is close to the theoretical one. We note that this speed can drop at simultaneous use of more than one high-performance interface because of the access to the same common memory.

The present paper proposes using the firmware systems obeying the HA model, and realization of such models in APSC.

The original concepts of this approach are presented in [19] and illustrated in Fig. 1 depicting the structure of the firmware system realized completely on one chip with the exception of part of the memory.

The software subsystem is realized in PS. If some operation needs acceleration, then the data for it are copied in the common memory which is accessible from PS and PL, and the hardware subsystem is activated. As was shown in [18], such activation can be executed most efficiently via the general-purpose (GP) ports. Memory access is done via the high-performance ports (see AXI HP and AXI ACP ports in [17, 18]). Upon operation completion, the PL generates an interrupt which serves as an indicator for PS. Completion of operation can be indicated by a signal transmitted via the general-purpose port. Numerous examples of the firmware interaction in APSC Zynq-7000 can be found in [20].

Control is exercised by a hierarchical automaton realized in part in software and in part in PL. The idea of such interaction was proposed in [19]. The programs in C/C++ use the conditional constructions like if ... else and switch ... case. The HA modules are described in terms of the language functions. The head module is realized in software and calls modules of lower level executed in the programs (in PS) and hardware (in PL). The hardware module is activated via the general-purpose port as is described in [10, 11], that is, the module name is transmitted via the port which allows one to select the necessary fragment of the algorithm. The design of hardware is done from the module descriptions very high speed integrated circuits hardware description language (VHDL). The desired fragment is selected in VHDL by the constructions case ... when. The corresponding methods are described in [21].

The firmware realization of HA is efficient and allows one to realize a simpler interaction of the HA modules controlling the hardware accelerators with signal transmitted by the chip contacts. Indeed, the PS is connected to the standard interfaces such as USB and UART. The interaction with external signals is done from PL where realized the hardware modules eliminating the need for additional devices for data exchange with the programs in PS.

For the basic problems requiring acceleration, consideration is given in what follows to the following operations of data analysis and processing that often occur in the control systems such as a) sorting, b) filtering, c) search, and d) counting. Sorting [22] orders data and underlies solution of many problems from items b–d. Filtering allows one to identify data within a given interval generated according to various criteria such as, for example, the Hamming weight [23]). Search implies determination of the maximum and/or minimum values [24], most frequently repeated values [25], maximum/minimum subsets [26], and solution of other similar problems. Counting allows one to determine some important characteristics of such data such as the number of nonzero values or the Hamming weight (HW), Hamming distance (HD), and so on. As will be shown, the efficient hardware solutions of such problems increasing significantly performance of the computer and control systems can be obtained by using the combinatorial networks for sorting and search [22, 24, 27] and enabling, if higher speed is required, the pipeline processing [27, 28]. The paper analyzes the existing works, proposes new approaches, and presents experimental results.

2. ANALYSIS OF THE EXISTING PUBLICATIONS

A detailed review of the parallel computations was given in [29], and numerous problem-oriented hardware accelerators. It was shown in [28, 30] that the combinatorial networks offer an efficient model of the hardware acceleration which can be widely used to solve the logic control problems such as [30, 31], data analysis and processing [22], and combinatorial optimization [32] (see, for example, the networks proposed in [33] for seeking covering Boolean matrices [32]). Numerous publications suggesting parallel sorting networks of various types such as the even-odd transition [34], even-odd merge [35, 36], bitonic merge [35, 36], bubble sort [37], insertion sort, and others are known. Such networks were analyzed and compared in [28]. The networks of even-odd and bitonic merge proposed in [35] are used most frequently as the hardware accelerators. It was shown in [27] that the networks [35] fail to operate in systems with high synchronizing frequency because of great

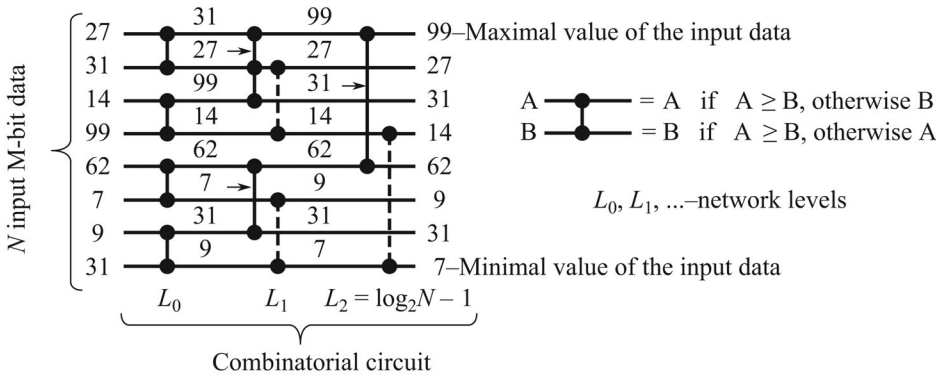

Fig. 2. Combinatorial network to determine the minimum and maximum of the input data.

combinatorial delay. The pipeline processing enables essential reduction of such delay, but gives rise to a new problem. The networks [35, 36] need many combinatorial comparators/switches and are unrealizable because of tremendous hardware. The iterative networks [27, 38] include parallel comparators/switches with the outputs connected to a register sending data to the inputs of the same comparators/switches until establishing the result of sorting. This approach enables one to reduce essentially the hardware and provide actually the same speed at the expense of an essentially shorter signal delay from the inputs to the outputs of the combinatorial part of the circuit. Table 1 shows the number of comparators $C(N)$ and circuit depth $D(N)$ for various methods of parallel sorting [27, 28], N being the number of the sorted data.

One more problem is related with transmission of large data volumes. Any network receives at its inputs N M-bit elements. The size of M often coincides with the standard bus size of 32 or 64 bits. The values of N usually exceeds hundreds. The number of data transmission channels usually is limited. For example, in the existing on-chip systems [15] the maximal number of high-speed channels (AXI HP and AXI AXP) is five. Consequently, large data volumes must be sent and transmitted in series which requires additional time delays often exceeding the data processing time.

The arising problems are readily eliminated in the network of [24]—see Fig. 2 where for clarity selected were particular values of the input data elements and $N = 8$—which after minor modification below can be used for sorting, search of maximum/minimum, and priority selection, which is often required in various control systems. Additionally, it is suggested to modify the network [24] so that the above tasks were executed in the course of data reception.

The network of Fig. 2 can be used to determine the maximal and minimal values of the input data. If the comparators shown by the dashed lines are removed, then determined is only the maximal value (call such network Max). Similarly, removal of the comparators shown by the arrows in Fig. 2 ($\rightarrow$) determines only the minimal value (call such network Min).

Table 1. Complexity $S(N)$ and speed $D(N)$ of various parallel networks

Type of sorting network	$C(N)$	$D(N)$
Bubble and insertion sorting	$N \times (N-1)/2$	$2 \times N - 3$
Even-odd transition	$N \times (N-1)/2$	N
Even-odd merge	$(p^2 - p + 4) \times 2^{p-2} - 1$, where $N = 2^p$	$p \times (p+1)/2$, where $N = 2^p$
Bitonic merge	$(p^2 + p) \times 2^{p-2}$, where $N = 2^p$	$p \times (p+1)/2$, where $N = 2^p$
Iterative sorting using two levels of even-odd transition [27]	$N - 1$	$\leq N$

The combinatorial networks are widely used not only for sorting. A parallel counting network proposed in [39] not only enables one to determine HW and HD, but also to compare HW. For the binary vectors, HW is the number of units in the vector, and such counting is widely used in the combinatorial search [32]. Numerous publications that are analyzed in [40] are devoted to the HW for different vectors. Numerous examples of practical use can be found in [7, 39]. A parallel network enabling one to determine the most frequently repeated values in a data set was proposed in [41]. Solution of such problem consists of two stages, first of which is sorting. Networks determining the maximal and minimal data subsets in a given set were proposed in [26]. Again, sorting underlies the proposed methods. In connection with the aforementioned, the networks [24] enabling one to simplify and accelerate solution of the corresponding problems are of interest for further consideration, realization, analysis, possible improvement, and subsequent comparison with other existing networks.

3. COMBINATORIAL NETWORK-BASED HARDWARE ACCELERATORS

On the basis of the network shown in Fig. 2, proposed are three types of accelerators that can be used for

1) determination of the maximal I_{max} and minimal values I_{min};

2) real-time sorting with data reception/transmission;

3) selection in real time of the most prioritized values.

The network shown in Fig. 2 is used unchanged to solve the problems of item 1. Item 3 requires construction of a priority buffer. Let Z_0, Z_1, Z_2, ... be a sequence of the M-bit input data beginning from the given Z_0. The aim of the problem is to select the most prioritized value, which can be either maximal of minimal, immediately after the arrival of some data and the most prioritized value must be selected from the already arrived data. Such problems often arise in the logic control systems (see, for example, [42, 43]). We assume for definiteness that the most prioritized value must be maximal. Then, for example, upon receiving the following data 66, 55, 99, 97, the result must be 99. Figure 3 depicts the proposed network which in this case is iterative. Each iteration is activated by a synchro pulse and corresponds to receiving a current value.

All registers are preloaded with the minimal values which must be smaller than the least-priority value. Let us assume that such values are zeros. The data are fed successively to the input of lower register. The most prioritized data having by assumption the maximal value is always moved to the upper register. After filling all registers, reception of the input data is stopped. Reading of the

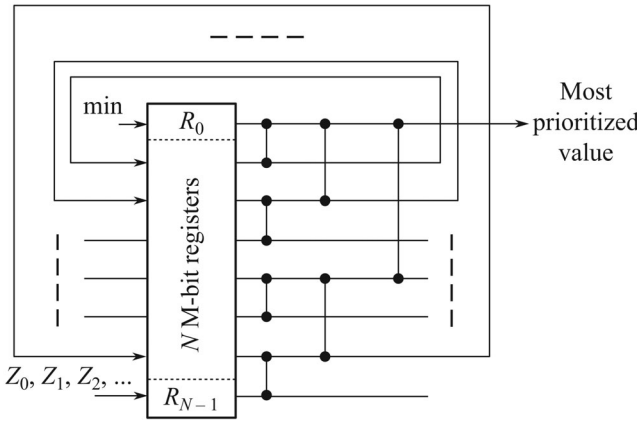

Fig. 3. Priority buffer.

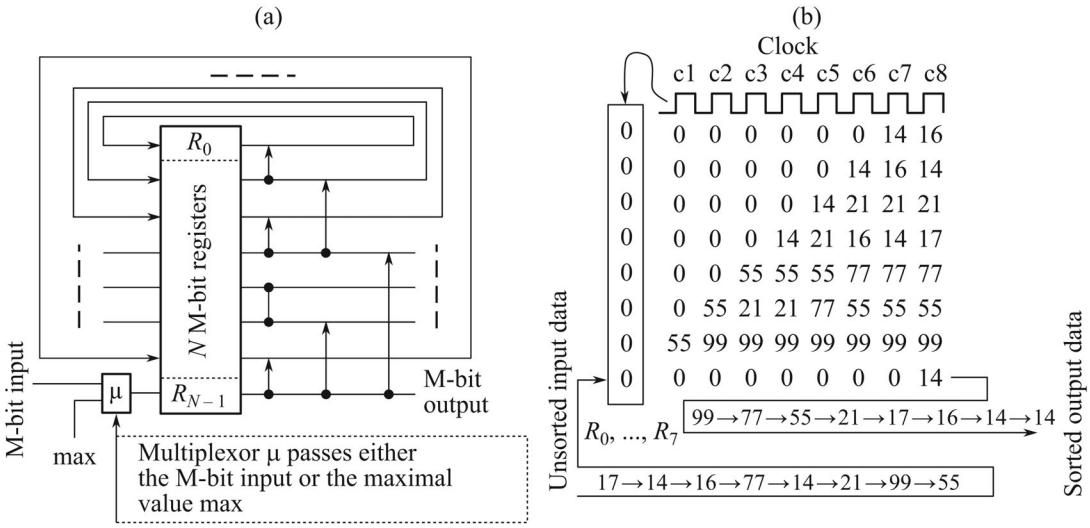

Fig. 4. (a) Network sorting data concurrently with their reception and (b) an example.

most prioritized value can be done at any time instant (the read data is removed and the minimal value min is written instead. At each cycle the value of min is moved to the lower register, and if all registers are not filled, this value is replaced by the input data.

The network sorting in real time (see above item 2) is shown in Fig. 4. At initialization all registers $R_0, \ldots, R_{N-1}$ are filled with the minimal possible values. As above, we assume that these are zeros.

Figure 4b shows how the arriving data are transformed by the network at each cycle of the synchronizing pulses s1, ..., s8. The network outputs are embraced by feedbacks, the register inputs where the intermediate results are stored at each iteration. In N cycles that are required for successive reception of all data, the sorted data can be successively transmitted from the M-bit output (see Fig. 4b). At that time the multiplexor μ transmits the maximal possible value of max which is moved at each cycle to the upper registers and permits output of the next sorted out value.

4. COMPARISON OF THE PROPOSED AND EXISTING COMBINATORIAL NETWORKS

Consider first the network in Fig. 2 having three combinatorial levels L_0, L_1, and L_2 and realizing the binary search of the maximal and/or minimal values. Since the search is binary, the number of such levels $D(N)$ for an arbitrary network is $\log_2 N$ which defines the conditional delay of signals passing from inputs to the outputs. For example, if $N = 1024$, then the delays for the network of Fig. 2 and for the fastest of the existing networks [35, 36] (see Table 1) are 10 and 55, respectively, that is, the network in Fig. 2 is 5.5 fold times faster. The number of elements (comparators/switches) of the network $S(N)$ at the level L_0 is $N/2 = N/2^1$. The number of elements at each successive level for the Min/Max networks varies as $N/2^2, \ldots, N/2^{\mathrm{p}}$, where $p = \log_2 N$. If both minimum and maximum are calculated simultaneously, then the number of elements of the level L_0 is still $N/2$, and the number of elements at the subsequent levels is $2(N/2^2, \ldots, N/2^{\mathrm{p}})$: half comparators are used to determine the minimum, and the second half, maximum. Table 2 gives formulas for computation of the conditional delay $D(N)$ and complexity $C(N)$ for the proposed networks.

Figure 5 shows theoretical comparison of the existing and proposed networks in terms of hardware such as comparators and switches. One can see that the proposed networks are the most economical.

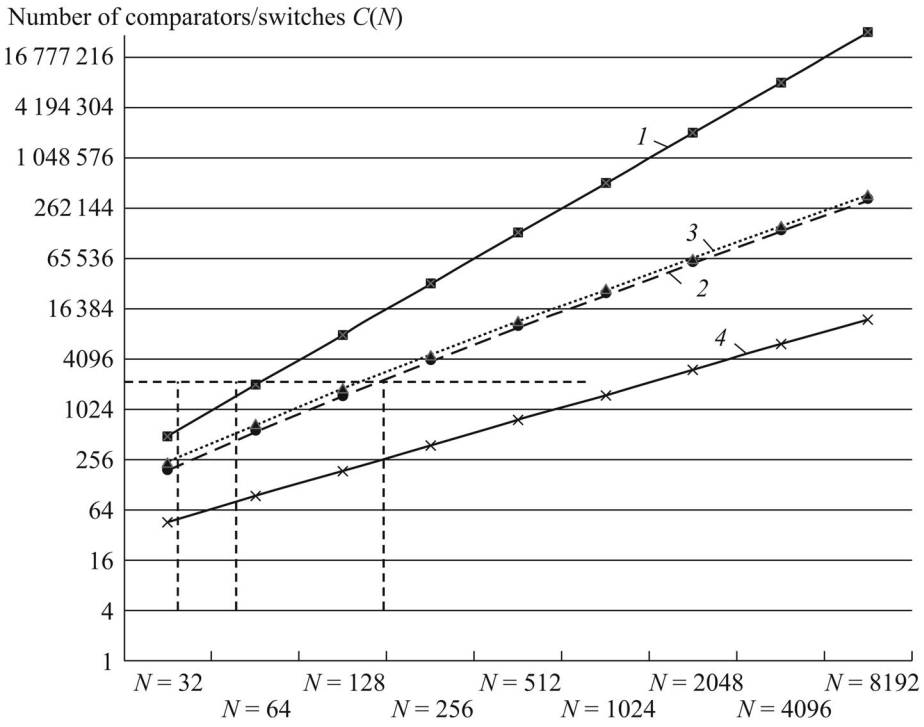

Fig. 5. Theoretical comparison of the existing and suggested conditions in terms of hardware costs: (*1*) bubble sorting, insertion sort, and even-odd transition, (*2*) even-odd merge, (*3*) bitonic merge, (*4*) proposed network (Min and Max).

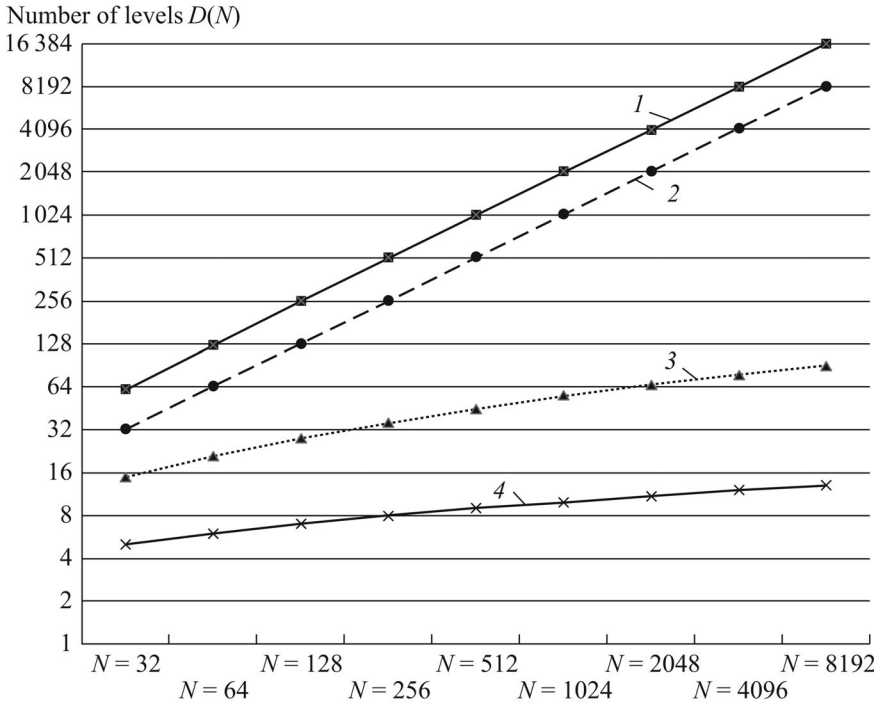

Fig. 6. Theoretical comparison of the existing and proposed networks in terms of speed: (*1*) bubble and insertion sorting, (*2*) even-odd transition, (*3*) even-odd and bitonic merge, (*4*) proposed network (Min and Max).

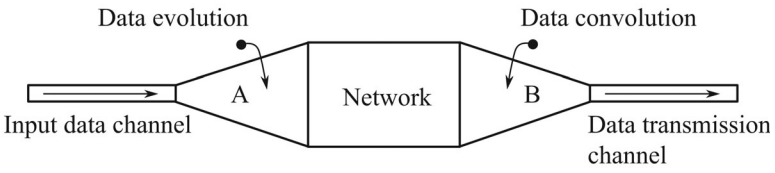

Fig. 7. Data reception and transmission via limited-size channels.

Figure 6 shows theoretical comparison of the existing and proposed networks in terms of the conditional speed (conditional delay of signal from input to output). Again, the proposed networks are the fastest.

The results of comparison are precise for the network of Fig. 2. The scheme for the network of Fig. 3 makes use only of the elements of Fig. 2 shown by the arrows $\rightarrow$. Consequently, for the same number of levels the hardware cost decreases as is shown in the "Min or Max" row of Table 2. A similar, that is, diminished, hardware costs has the scheme in Fig. 4 for the same number of levels as in the scheme of Fig. 2.

The proposed networks have one more advantage. Any of the existing networks (see Table 1) needs additional hardware and increases the hardware solution time. The necessary explanations are given in Fig. 7. The input data must be received via a limited number of data transmission channels. For example, it is impossible to get simultaneously 1024 32-bit data. In this case, the microchip must have either $1024 \times 32 = 32\,768$ external contacts or 32 768 internal interconnections, which is practically impossible. For example, the maximal number of internal interconnections for transmission of data in [17] through the high-speed interfaces is $5 \times 64 = 320$ and through the general-purpose ports—$4 \times 32 = 128$. The number of the external contacts of the most advanced chip of the Virtex-7 family is 1200 (chip XC7V2000T). Consequently, the input data must be received by serial segments, which requires additional time and resources. For many networks such as the sorting ones, the outputs also must be transmitted by segments. For example, in the case of sorting 1024 32-bit words one must organize successive transmission of groups of such words through a limited number of data transmission channels from the network to the memory or processor.

The complexity estimates and delays of signals for the components A and B (Fig. 7) can be obtained only for a particular system and not for a general case. For the system of [17], the hardware overhead is at least 10 % of the network complexity. Any existing network can process on the same hardware much less amount of data than the proposed network because the latter has much smaller amount of hardware (see Fig. 5). It follows that other advantages of the proposed networks such as the burst mode are realized much more efficiently. At sorting greater data arrays, the sorted out program blocks are usually merged [28], that is, realized is the merge sort method using the sorted blocks as the lower-level elements. The greater are the sorted blocks, the smaller merge levels are executed in the programs [27, 30]. Since the programs are essentially slower than the hardware, increase of block size leads to a substantial increase in the speed of sorting. Figure 5

Table 2. Complexity $S(N)$ and speed $D(N)$ of the proposed networks

Type of sorting network	$C(N)$	$D(N)$
Min or Max	$\sum_{n=1}^{\lceil \log_2 N \rceil} N/2^n$	$\lceil \log_2 N \rceil$
Min and Max simultaneously	$\sum_{n=1}^{\lceil \log_2 N \rceil} N/2^n + \sum_{n=1}^{\lceil \log_2 N \rceil} N/2^n$	$\lceil \log_2 N \rceil$

proves that for the same hardware charge (the same number of the comparators/switches used) the proposed networks sort much greater blocks. For example, for 1024 elements (see the horizontal bold dashed line in Fig. 5) the proposed networks enable one to sort eight-fold greater blocks than the best of the existing networks (compare N for various dashed vertical lines in Fig. 5).

5. REALIZATION OF THE FIRMWARE SYSTEM AND EXPERIMENTS

The networks shown in Figs. 2–4 were described in VHDL (the reader is referred to [30] for examples), designed in Xilinx Vivado 2015.2 system, and realized in the firmware complex (see Fig. 1) with the chips of the family Zynq-7000 [17]. The programs in S/S++ were developed in the Xilinx SDK 2015.2 system (SDK implies the Software Development Kit), loading the programs in PS, enables one to execute them on ARM Cortex-A9, that is, in PS, and interact with the hardware in PL. The data blocks with different number of elements (from 32 to 1024 32-bit elements) are generated in PS with the use of the rand function stored in the on-chip memory (OCM) [17] accessible from PS and PL via the high-speed ports. PL successively loads the block elements and processes the determined data. The results are transmitted either to the in-chip memory (OCM [17]) or immediately to PS. For example, at sorting large data arrays exceeding the block dimensions the sorted out blocks in PS are merged as described in [27, 28]. In all considered problems the priority buffer network (see Fig. 3) was not more than several hundred elements. It was constructed completely within PL, and the most prioritized elements were selected and transmitted through the general-purpose ports.

The experiments were carried out using two commercial boards ZyBo [44] with chip (with APSC) Zynq-7000 xc7z010 and ZedBoard [45] with chip Zynq-7000 xc7z020. Realized and compared were the proposed networks, iterative networks [27], and the network of even-odd merge [35, 36] of the firmware systems [28] that were used as the basis in the majority of the existing methods analyzed in Section 2 and have better theoretical characteristics together with the network [27] as is shown in Table 1. All components necessary for the hardware part including both the network and the data reception and transmission elements, that is, all resources used in PL were taken into consideration at counting the resources. Time was calculated for block sorting by PL on the maximal possible frequency which is lower for the networks [35, 36] than the proposed networks because of different depth of the combinatorial circuits (see Section 2 and Table 1). We notice that in the case of pipeline processing of numerous blocks the maximal permissible frequencies for all analyzed networks become equal, and in this case the hardware costs become the most significant factor.

Figure 8a shows the full time in nanoseconds of sorting one block without the pipeline processing. In all experiments, the data were processed sixty-four times by the network and then the mean value was used. The elapsed time consists of the sorting time and the times of data reception and transmission. The proposed network (see Fig. 4) is the fastest one and realizable in the ZedBoard for all sizes of the considered blocks. The existing network [35, 36] can be realized only for $N \le 64$ (ZedBoard) and $N \le 32$ (ZuBo). Figure 8b shows the number of used basic look-up tables (LUT) in PL. Such elements define the critical resource, that is, for the considered problems their number is always greater than the number of other elements such as the flip-flops, built-in memory, signal processing blocks. The condition [27] without parallelization of the input data and convolution of the output data requires less hardware then the proposed networks (see Fig. 8b). However, the resources required to get data from the input port and convolving the sorted results for their transmission through the output port increases the hardware costs, and without them the network [27] cannot be used in the firmware system. With regard for these resources, the proposed networks become most efficient (see Fig. 8b). Additionally, only they can be used for the prioritized selection in real time (see Fig. 3).

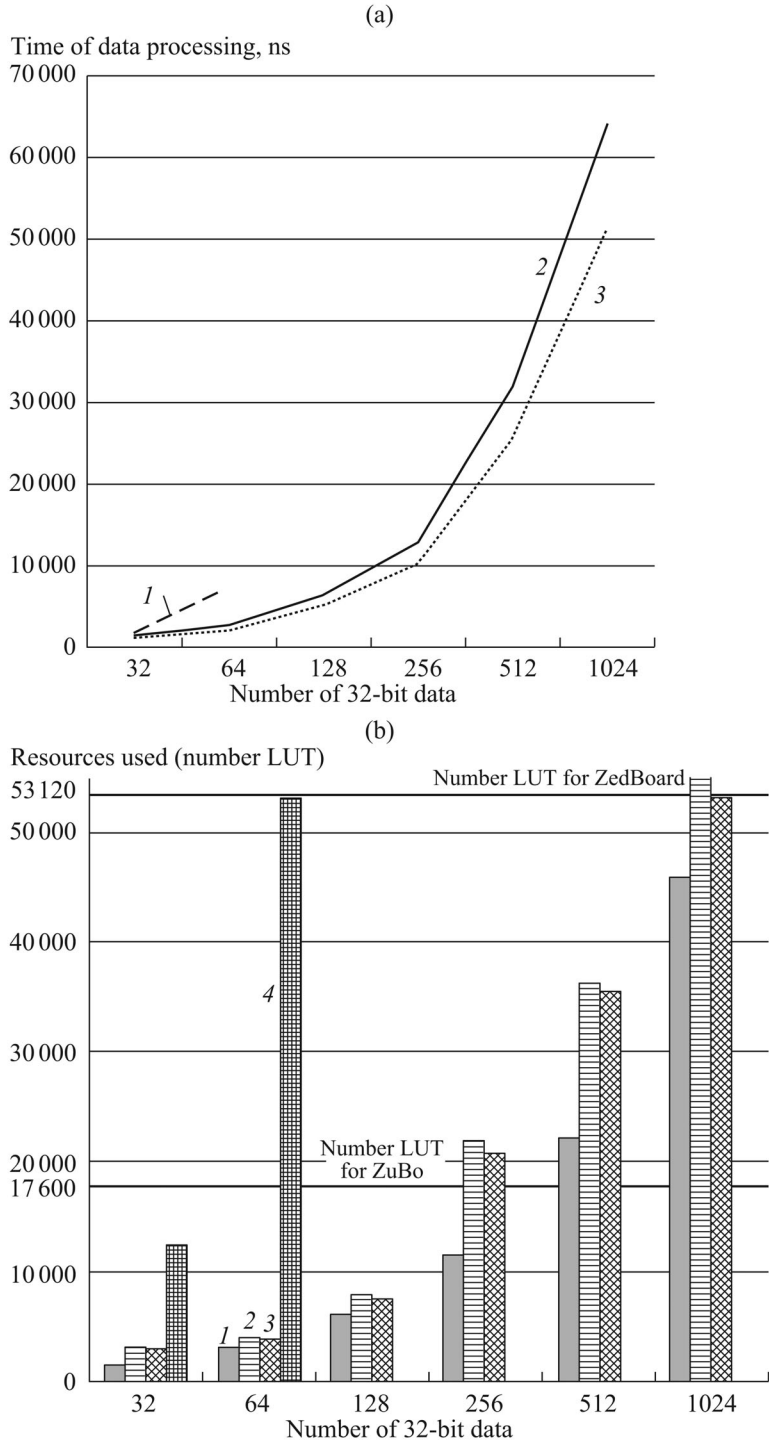

Fig. 8. (a) Time of data processing: (*1*) even-odd merge, (*2*) iterative networks [27], (*3*) proposed networks; (b) used resources: (*1*) iterative networks [27], (*2*) iterative networks [27] (network plus input and output resources), (*3*) proposed networks, (*4*) even-odd merge.

If multiple blocks are processed, then a pipeline can be used [28]. In this case, the throughput of the networks themselves becomes practically the same for all cases. However, the proposed networks again turn out to be faster because they process data concurrently with their arrival, which also was confirmed experimentally. Indeed, even if reception and sorting of blocks in the

existing networks are done concurrently, all the same one has to wait for complete copying of the first block and only then carry out sorting (see Fig. 7). In the proposed network (see Fig. 4) the result can be output immediately after getting the last block element. Moreover, the existing network [27] requires more involved timing and control for pipeline processing, which results in an additional amount of the resources used.

The networks [35, 36] can be used only for the blocks with small number of elements.

The calculation problems complement the proposed methods, and solution of such problems is determined using the existing combinatorial networks.

6. CONCLUSIONS

Proposed were fast combinatorial networks for different types of data processing such as sorting, search, and filtration. Their efficiency in the firmware systems and their advantages (lower hardware costs and higher speed) in comparison with the existing alternatives were demonstrated. Efficiency of using the proposed networks in the control systems, as well as for data analysis and processing was demonstrated.

REFERENCES

1. Lee, E.A. and Seshia, S.A., *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, Berkeley, 2015, 2nd ed.

2. Jensen, J.C., Lee, E.A., and Seshia, S.A., *An Introductory Lab in Embedded and Cyber-Physical Systems*, Berkeley, 2015.

3. Rajkumar, R., Lee, I., Sha, L., et al., Cyber-Physical Systems: The Next Computing Revolution, in *Proc. 47th ACM/IEEE Design Automation Conf.*, Anaheim, California, 2010, pp. 731–736.

4. Vipin, K., Shreejith, S., Fahmy, S.A., et al., Mapping Time-Critical Safety-Critical Cyber Physical Systems to Hybrid FPGAs, in *Proc. 2nd IEEE Int. Conf. on Cyber-Physical Systems, Networks, and Applications*, IEEE Computer Society, 2014, pp. 31–36.

5. Panunzio, M. and Vardanega, T., An Architectural Approach with Separation of Concerns to Address Extra-functional Requirements in the Development of Embedded Real-time Software Systems, *J. Syst. Architect.*, 2014, vol. 60, no. 9, pp. 770–781.

6. Borangiu, A. and Popescu, D., Digital Signal Processing for Knowledge Based Sonotubometry of Eustachian Tube Function, *J. Control Eng. Appl. Inform.*, 2014, vol. 16, no. 3, pp. 56–64.

7. Sklyarov, V. and Skliarova, I., Digital Hamming Weight and Distance Analysers for Binary Vectors and Matrices, *Int. J. Innovat. Comput., Inform. Control.*, 2013, vol. 9, no. 12, pp. 4825–4849.

8. Benmoussa, Y., Boukhobza, J., Senn, E., et al., A Methodology for Performance/Energy Consumption Characterization and Modeling of Video Decoding on Heterogeneous SoC and its Applications, *J. Syst. Architect.*, 2015, vol. 61, pp. 49–70.

9. Zmaranda, D., Silaghi, H., Gabor, G., et al., Issues on Applying Knowledge-Based Techniques in Real-Time Control Systems, *Int. J. Comput., Commun. Control*, 2013, vol. 8, no. 1, pp. 166–175.

10. Sklyarov, V.A., *Sintez avtomatov na matrichnykh BIS* (Design of Automata on Matrix VLSI) Minsk: Nauka i Tekhnika, 1984.

11. Sklyarov, V., Hierarchical Finite-State Machines and Their Use for Digital Control, *IEEE Trans. VLSI Syst.*, 1999, vol. 7, No. 2, pp. 222–228.

12. Sklyarov, V., Reconfigurable Models of Finite State Machines and their Implementation in FPGAs, *J. Syst. Architect.*, 2002, vol. 47, pp. 1043–1064.

13. Baranov, S.I. and Sklyarov, V.A., *Tsifrovye ustroistva na programmiruemykh BIS s matrichnoi strukturoi* (Digital Devices on Programmable LSI), Moscow: Radio i Svyaz', 1986.

14. Sklyarov, V. and Skliarova, I., Hardware Implementations of Software Programs Based on HFSM Models, *Comput. Electr. Eng.*, 2013, vol. 39, no. 7, pp. 2145–2160.

15. Santarini, M., Products, Profits Proliferate on Zynq SoC Platforms, *XCell*, 2014, no. 88, pp. 8–15.

16. Santarini, M., Xilinx 16nm UltraScale+ Devices Yield 2-5X Performance/Watt Advantage, *XCell*, 2015, no. 90, pp. 8–15.

17. Xilinx, Inc., *Zynq-7000 All Programmable SoC Technical Reference Manual*, 2014, http://www.xilinx.com/ support/documentation/user_guides/ug585-Zynq-7000-TR M.pdf.

18. Silva, J., Sklyarov, V., and Skliarova, I., Comparison of On-chip Communications in Zynq-7000 All Programmable Systems-on-Chip, *IEEE Embedded Syst. Lett.*, 2015, vol. 7, no. 1, pp. 31–34.

19. Sklyarov, V.A., Microprocessor Device for Control of Industrial Equipment, *Autom. Remote Control*, 1985, vol. 46, no. 1, pp. 102–105.

20. Sklyarov, V., Skliarova, I., Silva, J., et al., *Hardware/Software Co-design for Programmable Systems-on-Chip*, Tallinn: TUT Press, 2014.

21. Skliarova, I., Sklyarov, V., and Sudnitson, A., *Design of FPGA-based Circuits Using Hierarchical Finite State Machines*, Tallinn: TUT Press, 2012.

22. Knuth, D.E., *The Art of Computer Programming*, vol. 3: *Sorting and Searching*, Reading: Addison-Wesley, 2011.

23. Pedroni, V., Compact Hamming-comparator-based Rank Order Filter for Digital VLSI and FPGA Implementations, in *Proc. IEEE Int. Symp. on Circuits and Syst.*, 2004, vol. 2, pp. 585–588.

24. Sklyarov, V. and Skliarova, I., Fast Regular Circuits for Network-based Parallel Data Processing, *Adv. Electr. Comput. Eng.*, 2013, vol. 13, no. 4, pp. 47–50.

25. Teubner, J., Mueller, R., and Alonso, G., Frequent Item Computation on a Chip, *IEEE Trans. Knowledge Data Eng.*, 2011, vol. 23, no. 8, pp. 1–15.

26. Sklyarov, V., Skliarova, I., Rjabov, A., et al., Zynq-based System for Extracting Sorted Subsets from Large Data Sets, *J. Microelectron., Electron. Components Mater.*, 2015, vol. 45, no. 2, pp. 142–152.

27. Sklyarov, V. and Skliarova, I., High-performance Implementation of Regular and Easily Scalable Sorting Networks on an FPGA, *Microprocessors Microsyst.*, 2014, vol. 38, no. 5, pp. 470–484.

28. Mueller, R., Teubner, J., and Alonso, G., Sorting Networks on FPGAs, *Int. J. Very Large Data Bases*, 2012, vol. 21, no. 1, pp. 1–23.

29. Bunich, A.L., Ginsberg, K.S., Dobrovidov, A.V., Zatuliveter, Yu.S., Prangishvili, I.V., Smolyaninov, V.V., and Sukhov, E.G., Parallel Computation and Control Problems: A Review, *Autom. Remote Control*, 2002, vol. 63, no. 12, pp. 1867–1883.

30. Sklyarov, V., Skliarova, I., Barkalov, A., et al., *Synthesis and Optimization of FPGA-based Systems*, New York: Springer, 2014.

31. Zakrevskij, A. and Sklyarov, V., The Specification and Design of Parallel Logical Control Devices, in *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2000, pp. 1635–1641.

32. Zakrevskii, A.D., *Logicheskii sintez kaskadnykh skhem* (Logical Design of the Cascade Circuits), Moscow: Nauka, 1981.

33. Sklyarov, V., Skliarova, I., Rjabov, A., et al., Fast Matrix Covering in All Programmable Systems-on-Chip, *Electron. Electric. Eng.*, 2014, vol. 20, no. 5, pp. 150–153.

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

35. Batcher, K.E., Sorting Networks and Their Applications, in *Proc. AFIPS Spring Joint Computer Conf.*, 1968, pp. 307–314.

36. Aj-Haj Baddar, S.W. and Batcher, K.E., *Designing Sorting Networks. A New Paradigm*, New York: Springer, 2011.

37. Chamberlain, R.D. and Ganesan, N., Sorting on Architecturally Diverse Computer Systems, in *Proc. 3rd Int. Workshop on High-Performance Reconfigurable Computing Technology and Appl.*, 2009, pp. 39–46.

38. Zuluada, M., Milder, P., and Puschel, M., Computer Generation of Streaming Sorting Networks, in *Proc. 49th Design Automation Conf.*, 2012, pp. 1245–1253.

39. Sklyarov, V. and Skliarova, I., Design and Implementation of Counting Networks, *J. Comput.*, 2015, vol. 97, no. 6, pp. 557–577.

40. Parhami, B., Efficient Hamming Weight Comparators for Binary Vectors Based on Accumulative and Up/Down Parallel Counters, *IEEE Trans. Circuits Syst. II: Express Briefs.*, 2009, vol. 56, no. 2, pp. 167–171.

41. Sklyarov, V., Skliarova, I., and Kabulov, A.V., Finding Most Frequently Repeated Data in the Sorted Arrays, *Dokl. Uzbek Akad. Nauk*, 2014, no. 4, pp. 16–18.

42. Sklyarov, V., Skliarova, I., and Neves, A., Modeling and Implementation of Automatic System for Garage Control, in *Proc. ICROS-SICE Int. Joint Conf.*, 2009, pp. 4295–4300.

43. Sklyarov, V. and Skliarova, I., Modeling, Design, and Implementation of a Priority Buffer for Embedded Systems, in *Proc. 7th Asian Control Conf.*, 2009, pp. 9–14.

44. *Digilent, Inc. ZyBo Reference Manual*, http://digilentinc.com/Data/Products/ZYBO/ZYBO_RM_B_V6.pdf. 2014.

45. *Avnet, Inc. ZedBoard (ZynqTM Evaluation and Development) Hardware User's Guide, Version 2.2*, http://www.zedboard.org/sites/default/files/, 2014.

This paper was recommended for publication by V.M. Vishnevskii, a member of the Editorial Board