Hardware Accelerators for Data Sort in All Programmable Systems-on-Chip

Valery SKLYAROV, Iouliia SKLIAROVA
University of Aveiro/IEETA, 3810-193 Aveiro, Portugal
skl@ua.pt, iouliia@ua.pt

***Abstract*—The paper analyzes and evaluates architectures of the most efficient hardware accelerators for data sort in FPGA and all programmable systems-on-chip (such as devices from the Xilinx Zynq-7000 family). The following novel methods are proposed and discussed: 1) data sorting in hardware that is executed concurrently with getting inputs through single or multiple ports; 2) a technique allowing rational compromise between the cost and the latency of the circuit to be achieved. Both methods are targeted to hardware/software co-design and permit the best solution to be found for different requirements within pre-defined constraints. The results of experiments, implementations, and rigorous comparisons demonstrate high efficiency and broad applicability of the proposed methods for wide range of practical applications.**

***Index Terms*—FPGA, System-on-chip, Sorting, Parallel processing, Performance and resources evaluation.**

I. INTRODUCTION

All Programmable Systems-on-Chip (APSoC) from the Zynq-7000 family [1,2] combine on the same microchip the dual-core ARM® Cortex™ MPCore™-based high-performance processing system (PS) with advanced programmable logic (PL) from the Xilinx 7th family and may be used effectively for the design of hardware accelerators in such areas as hard real-time systems [3], image [4] and data [5] processing, satellite on-board processing [6], programmable logic controllers [7], driver assistance applications [8], wireless networks [9], and many others [2]. Interactions between the PS and PL are supported by different interfaces and other signals through over 3,000 connections [1]. The available four 32/64-bit high-performance (HP) Advanced eXtensible Interfaces (AXI) and 64-bit AXI Accelerator Coherency Port (ACP) enable fast data exchange with theoretical bandwidths reported in [1] and practical results shown in [10].

The design flow for Zynq APSoC includes development of hardware in the PL [11] (supported by available Xilinx IP cores) and software in the PS [12] for different types of applications such as standalone (bare metal) [13], running under an operating system (*e.g.* Linux) [13], and combined [14]. Hardware implemented in the PL can be the same for standalone and Linux applications but software programs use different functions and interaction mechanisms [13].

Sorting is a procedure that is needed in numerous computing systems [15]. For many practical applications, sorting throughput is very important. To better satisfy performance requirements, fast accelerators based on field-programmable gate arrays (FPGA) (*e.g.* [16-19]), graphics processing units (*e.g.* [20-21]) and multi-core central processing units (*e.g.* [22-23]) have been proposed. APSoC devices may combine the mentioned above accelerators taking advantage of the built-in high-performance PS and optimized circuits implemented in the PL.

The majority of known hardware accelerators for data sort use Batcher even–odd and bitonic mergers [24,25] which are the fastest because of the lowest latency L(N) measured by the number of levels of basic network elements through which signals propagate from the inputs to the outputs. Such elements are comparators/swappers for data items. Let $p = \log_2 N$, where N is the number of K-bit data items that have to be sorted. It is known that L(N) for both referenced above mergers [24,25] is equal to $p\times(p+1)/2$. The cost C(N) (the number of comparators/swappers) for the even-odd merger is smaller than for the bitonic merger and it is equal to $(p^2-p+4)\times 2^{p-2}-1$.

A review of recent results in hardware accelerators for data sort can be found in [26] which demonstrate that the resources available even in modern reconfigurable microchips only allow circuits to be constructed that can handle a very limited number N of items because of large values of C(N). It should be also noted that although methods [24,25] offer the best theoretical throughput, the actual performance is limited by the interfacing circuits that supply initial data and return the results. Indeed, even for the most recent and advanced on-chip interaction methods, such as that are used in APSoC [1], the communication overheads do not allow the theoretical throughput [24,25] to be achieved in practical designs [26]. It is shown in [10] that data exchange between the PS and PL involves delays and the bottleneck is in communications. Thus, executing sorting operations as soon as a new data item arrives might be useful and promising. Indeed, although even-odd and bitonic merges do not involve sequential operations, additional time and components are needed such as those to prepare long size input vectors and to transmit long size output vectors through interfaces with a limited number of lines (such as that are available for interactions between the PS and PL in APSoC [1]).

We describe below highly parallel networks that enable sorting to be done either entirely within the time required for data transfers to and from the circuit or with a minimal addition time. We will call such circuits either *communication-time* or *real-time*. It will be also shown that although minimizing the latency L(N) and the cost C(N) cannot be done simultaneously, some compromises between L(N) and C(N) may be found allowing either the fastest or the less resource consuming circuits to be designed which

This research was supported by 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.

Digital Object Identifier 10.4316/AECE.2015.04002

depends on the requirements and constraints.

The remainder of the paper is organized in 5 sections. Section II briefly discusses methods and related work. Section III suggests hardware accelerators that enable sorting to be done in real time with data exchange for supplying inputs and outputs. Section IV suggests circuits that permit the maximum number of data items to be processed within the given constraints for hardware resources. Section V describes experiments in FPGA and APSoC and reports the results of comparisons, clearly explaining why the proposed circuits are better than the known alternatives. Conclusion is given in section VI.

II. Methods and Related Work

The following two methods are the most commonly applied for sorting large data sets in software/hardware systems: a) large data sets are sorted in host computers/ processors through merging sorted subsets produced by an FPGA (see, for example, [16]); b) sorting networks for large sets are segmented in such a way that any segment can be processed easily and the results from the processing are handled sequentially to form the sorted set (see, for example, [17,21]). Both methods involve intensive communications, either between an FPGA and host computing system/external memory (the size of memory embedded to FPGA is limited), or between a processing system (such as [21]) and memory.

Fig. 1 outlines the basic architecture of hardware/software data sorters from [26] that will be used in this paper.

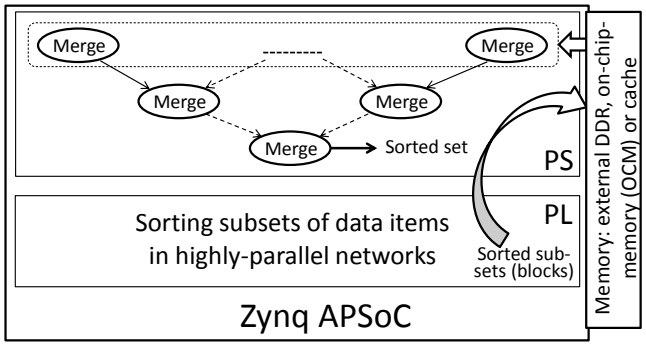

Figure 1. The basic architecture of the hardware/software data sorter

Large sets of data items are decomposed in subsets that can be sorted in the PL. We found that merging in software is significantly slower than sorting subsets in hardware with the aid of the methods [26]. Thus, on the one hand, to increase throughput of hardware/software data sorters we need to handle bigger subsets in the PL because the PL, executing many parallel operations, is expected to be faster. On the other hand, processing larger subsets in the PL may lead to performance degradation because it is usually done at the expense of decreasing the number of parallel operations and consequently the PL may become slower than the PS. Indeed, clock frequency of the PS is notably higher than clock frequency of the PL [1] and high-level parallelism in the PL must be provided to execute operations faster than in the PS. Another problem is communication overhead [10]. Copying data items from/to memories may be combined with solving other tasks in software in parallel. However, such combination is very questionable for the PL. Indeed, we would like to use as much hardware as possible to increase the size of sorted subsets. Thus, as a rule, no additional hardware is available in the PL for solving other problems in parallel. It is rather more efficient to find a way that enables sorting to be combined with data transfer. Section III below demonstrates that such a way is achievable.

III. Communication-Time Hardware Accelerators

There are five HP AXI ports between the PS (memories) and PL each of which can be configured for 32 or 64-bit data transfers. The theoretical bandwidth for read/write operations through one HP AXI port is 1,200 MB/s [1]. Dependently on requirements, data may be transferred through either a single or multiple ports.

III.1 Transferring Data through a Single Port

The proposed network, which is based on the circuit for discovering the minimum and maximum values from [27], is shown in Fig. 2. It is composed of N K-bit registers $R_0,\ldots,R_{N-1}$, and N-1 comparators/swappers. For the sake of simplicity, N is assigned to be 16. Clearly, other values may be chosen.

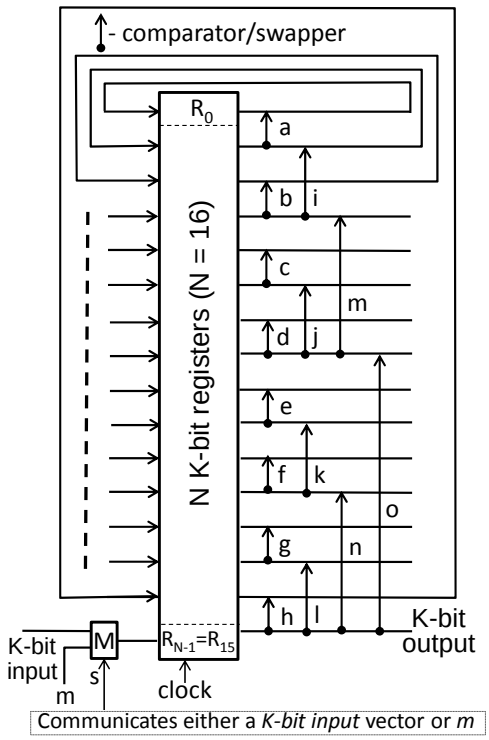

Figure 2. Real-time accumulator/sorter for N = 16

At the initialization step, all the registers $R_0,\ldots,R_{N-1}$ are set to the minimum possible value for data items. For the examples below we assume that this value is 0. Data items are received sequentially from interfacing circuits through the multiplexer M. Since all the registers are set to the minimum values, all input items with non-minimum values will be moved up and accommodated somehow in the registers $R_0,\ldots,R_{N-1}$. Fig. 3 demonstrates how N=16 K-bit items are accommodated using an example with data arriving in the following sequence: 1) 28; 2) 14; 3) 37; 4) 65; 5) 11; 6) 14; 7) 19; 8) 71; 9) 0; 10) 69; 11) 14; 12) 41; 13) 71; 14) 22; 15) 70; 16) 7. The circuit in Fig. 2 composed

of comparators/swappers is combinational and all the comparators/swappers a,…,o operate in parallel handling input data from the registers $R_0,\ldots,R_{N-1}$. Outputs of the circuit composed of comparators/swappers are written to the registers $R_0,\ldots,R_{N-1}$ through feedback connections and only the bottom output (marked as K-bit output) does not have a feedback. Note that data may be received from the PS (from memory) and accommodated in the registers $R_0,\ldots,R_{N-1}$ in N clock cycles indicated in Fig. 3 by symbols c1,…,c16 (N=16). As soon as N unsorted data are received, the sorted result can be transferred immediately to the PS as shown in Fig. 4.

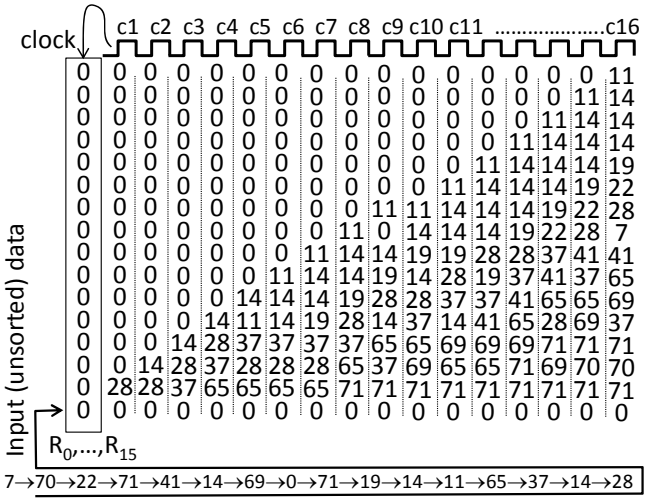

Figure 3. Iterations for acquisition of data items

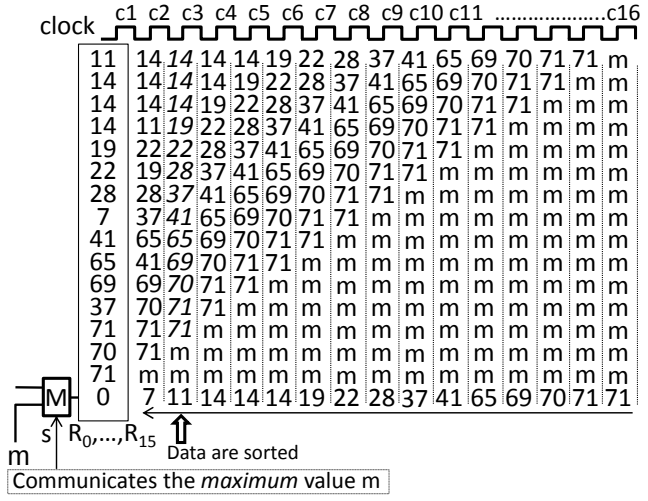

Figure 4. Transmitting the sorted data items

Let us look at Fig. 3. At each step, when a new item is received, the previously accommodated in the registers $R_0,\ldots,R_{N-1}$ items become partially sorted. This is because the network [27] provides for necessary data exchanges in the registers with the aid of comparators/swappers. Almost from the beginning of transmitting sorted data (see Fig. 4) all other data items in the registers become completely sorted (see the column shown in *italic font* and pointed by an "up arrow" with the message "Data are sorted"). It is done after the clock cycle c2 (the upper value 14 is the smallest and the bottom value 71 is the largest). This is because the proposed network always moves the maximum value m to upper positions. Thus, sorting is completed almost immediately after all input data have been transferred from a single port to the proposed circuit. Hence, outputs can be delivered to the PS (to memory) either through a single port as shown in Fig. 4 or through Q>1 ports using additional multiplexers selecting segments from the sorted items sequentially (see section V for additional details). For example, for Q=5, beginning from the clock cycle c3 the following three segments can be transmitted in cycles c3, c4, and c5: 1) 11, 14, 14, 14, 19; 2) 22, 28, 37, 41, 65; 3) 69, 70, 71, 71. Since in the first two cycles c1 and c2 the values 0 and 7 have already been transmitted, the total number of clock cycles is just 5. Note, that this method requires reconfiguration of input/output (I/O) ports in the PS which might involve more time than transferring data items through a single port. Analysis of software and hardware capabilities permits the best solution to be chosen.

III.2 Transferring Data through Multiple Ports

Fig. 5 depicts the proposed network for Q ports. The circuit is decomposed into Q autonomous sub-circuits (segments) of equal size ξ (i.e. up to ξ items can be acquired by each sub-circuit). The network in Fig. 5 assumes Q=4. Clearly, other values may be chosen. Each segment can handle an arbitrary number of data items (provided sizes ξ of all segments are equal). The only requirement is that any segment finds an item with the minimum value.

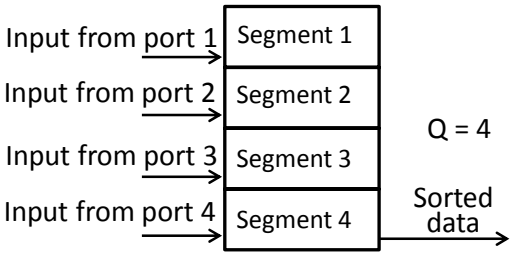

Figure 5. Transferring data items through multiple ports

Data items are received in parallel from Q ports in such a way that port i supplies inputs for the segment i (i = 1,…,Q). Any segment is a circuit shown in Fig. 2. Different segments are linked by comparators/swappers. Swapping of data items between different segments cannot occur at any parallel data transfer (of up to ξ sequential items) through Q input ports except the last one for which such swapping is indeed needed to deliver the smallest item. As soon as all input data are saved in the registers $R_0,\ldots,R_{N-1}$ exactly the same functionality as in Fig. 4 is provided. Note that more clock cycles than in Fig. 4 will be needed to sort all the items. Completion of sorting may be recognized by additional comparators verifying that any upper item is smaller than or equal to a neighboring lower item.

Fig. 6 presents an example of the circuit in Fig. 2 divided in 4 segments (Q = 4). A segment may have a different number of lines (i.e. not obligatory a power of 2), but we have already mentioned that the bottom line of any segment has to always contain the smallest value. Note that only lines with the smallest values participate in potential data swapping between the segments. Clearly, for all data transactions 0,1,…,ξ-2 the bottom lines in all segments contain the same smallest value (that is the predefined

minimum such as 0). Thus, although comparators/swappers (such as n, m, o in Fig. 6) connect different segments there is no need for swapping of data. For the last (ξ-1) data transfer the smallest values in bottom lines of the segments will be replaced with other values. Now, swapping between segments may occur and it permits to deliver the smallest value to the bottom line of the network composed of 4 segments. Hence, transfer of the sorted items can be done immediately.

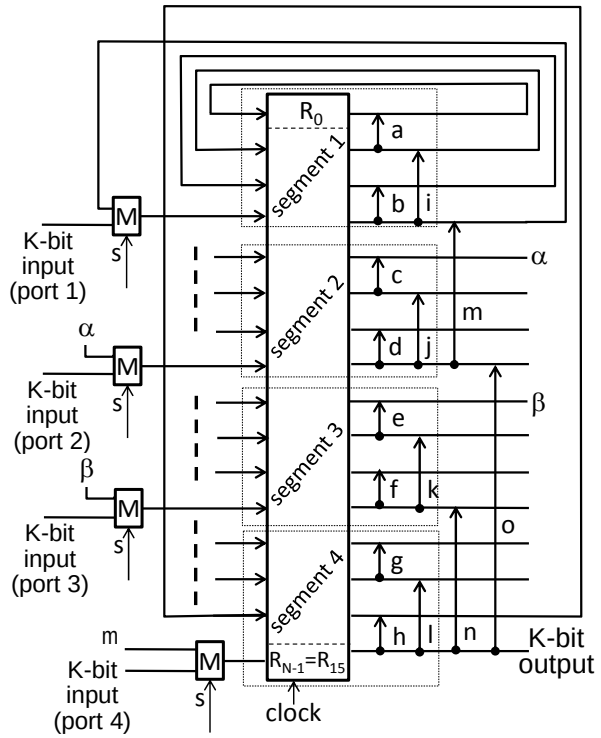

Figure 6. Using 4 ports for input data and 1 port for output data

Fig. 7 gives the same example as in Fig. 3, 4 but uses Q=4 input ports instead of just one. Data are acquired in 4 clock cycles, i.e. 4 times faster than in Fig. 3 (where data are received in 16 clock cycles). Data get sorted in 11 clock cycles (from c5 to c15). So, the total throughput is higher than for Fig. 4.

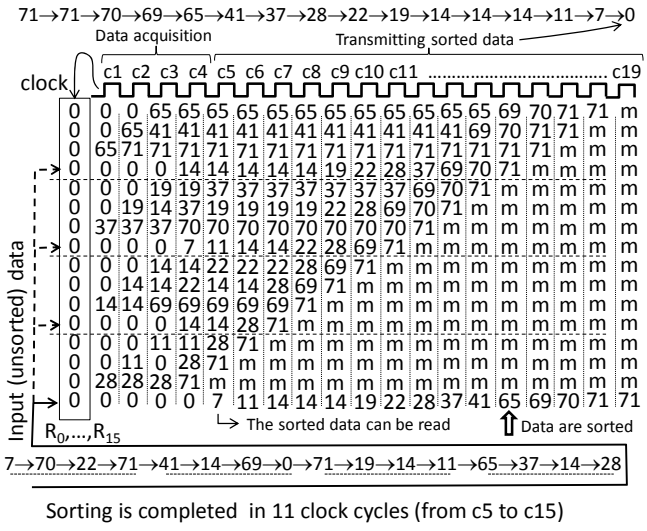

Figure 7. An example of data acquisition and sort for Fig. 6

If input ports will further be used as output ports, the PS has to reconfigure them from reading data (i.e. PS→PL) to writing data (i.e. PL→PS) and this involves additional time. Our experience has shown that such additional time is not less than the extra delay for data sort. Therefore, we can again consider the proposed sorting as communication-time. Alternatively, data items may be acquired through 4 ports and delivered through the fifth port without reconfiguration.

IV. Processing Larger Sets of Data in the PL

Thorough experiments with data sorters from previous sections (the results will be reported in section V) have demonstrated that although the proposed networks are very fast they do not permit larger circuits than in [26] to be built within resource constraints of the given PL. We found that data sorters for larger sets have to be as regular as possible.

Let us look at the circuit in Fig. 8 that is composed of N K-bit registers $R_0,\ldots,R_{N-1}$ with comparators/swappers between them. Any comparator/swapper compares items in upper and bottom registers and transfers the item with larger value to the upper register (let us call it A) and the item with smaller value to the bottom register (let us call it B). Thus, if A < B then data in the registers are swapped, otherwise they are unchanged. Such operations are applied simultaneously to all the registers connected to even comparators/swappers (0, 2, 4,…) in one clock cycle (let us call it even clock cycle) and to all the registers $R_0,\ldots,R_{N-1}$ connected to odd comparators/swappers (1, 3, 5,…) in a subsequent clock cycle (let us call it odd clock cycle).

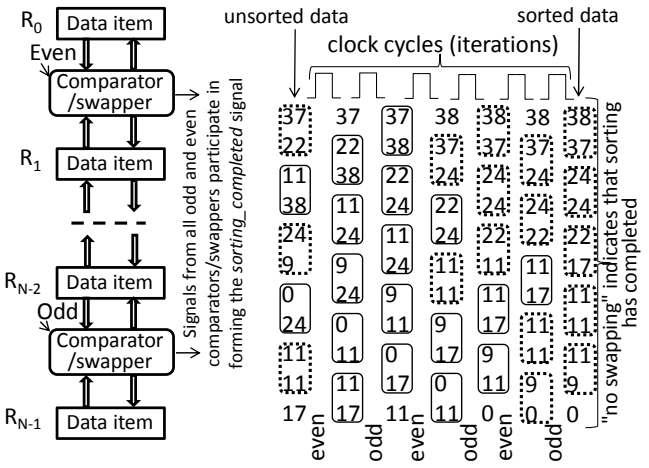

Figure 8. Sorting circuit with an example

Clearly, this implementation can be unrolled to the combinational even-odd transition network [20], but such a network requires significantly larger number of comparators/swappers that is equal to N×(N-1)/2. Since we execute not only combinational but also sequential operations with the comparators/swappers we will call our circuit *iterative even-odd transition network*. Note that this network is not the same as in [26] because any comparator/swapper is dedicated to the relevant pair of registers and all the required interconnections are very simple and easily implementable. Evidently, the maximum number of iterations is equal to N but it may be reduced much similarly to [26]. Indeed, if beginning from the second iteration there is no data exchange in all either even or odd comparators/swappers then the data items are sorted. At the first iteration if there is no data swapping, data swaps for the

next iteration still may occur. We assume that the first iteration always involves even comparators/swappers.

Let us look at the example shown in Fig. 8 (N=11). At the beginning, unsorted data are copied to the registers $R_0,\ldots,R_{N-1}$. Each iteration (6 iterations totally) is forced by clock edge. Rounded dotted and solid rectangles in Fig. 8 indicate elements that are compared in iterations 1-6. Rounded solid rectangles enclose data items that are actually swapped. Data are sorted in 6 clock cycles and 6 < N=11.

The main disadvantage of the circuits from this section comparing to section III is the necessity to copy long size sets to/from the registers $R_0,\ldots,R_{N-1}$ (before the sorting starts). However, the ideas from section III may also be applied to the circuit in Fig. 8. As can be seen from Fig. 9 data items can be copied from a port in even clock cycles and subsequent sorting is done in smaller number of clock cycles after data acquisition than in Fig. 8. Besides, the circuit does not require additional components to fill in a long size input register (needed for $R_0,\ldots,R_{N-1}$). Our experiments have shown that although the circuit in Fig. 9 is not as fast as the circuits in section III, it occupies less PL resources than the circuits from section III due to higher regularity.

A similar communication-time technique may also be used for known iterative even-odd transition networks from [5,26]. Indeed, a K-bit input vector can be delivered to the bottom line of the circuit from [5,26] much like it is done in Fig. 9. We think that the number of additional clock cycles (that are needed beyond data transfers) is smaller in circuits [5,26] but due to the higher regularity of the circuit in Fig. 9, a larger sorter might be implemented within the same available resources. Experiments will be done in section V.

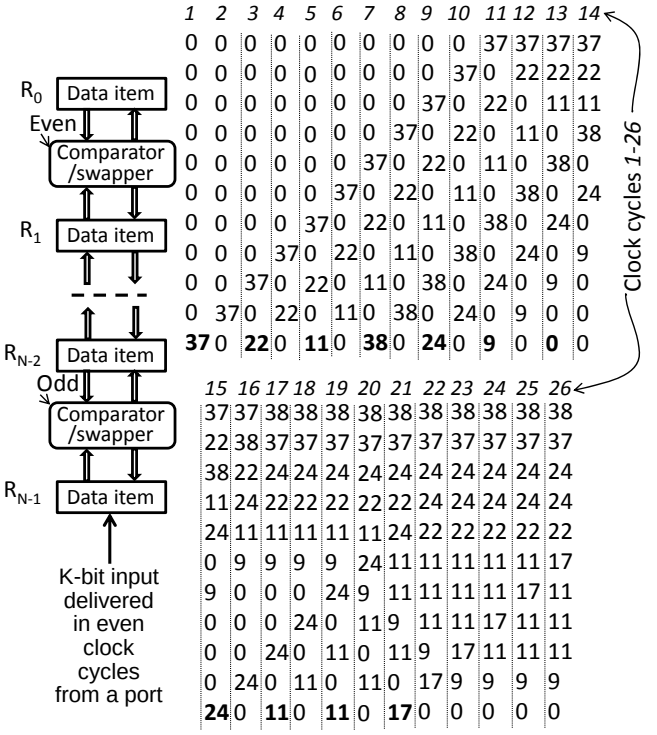

Figure 9. Communication-time data acquisition and sorting for Fig. 8

V. IMPLEMENTATIONS, EXPERIMENTS, AND COMPARISONS

Different data sorters can be characterized by the following parameters:

- The number L(N) of combinational levels;
- The number η(N) of iterations required for sorting the given set from N items in iterative circuits;
- The cost C(N) that is the number of comparators/ swappers;
- The maximum attainable clock frequency F_{max} for iterative circuits;
- Throughput τ that is the number of items sorted in time unit (such as a second).

Table I gives the values L(N), η(N), and C(N) for the known and the proposed circuits that can be found from theoretical expressions presented above in the paper and in [5,16,20,26]. The values of η(N) are given only for iterative circuits (because they are not needed for pure combinational circuits). Indices 2, 8, and 9 in Table I indicate the values for Fig. 2, 8, and 9. Other indices refer to the following sorters:

- *eom* – even-odd merge [24,25];
- *b* – bitonic merge [24,25];
- *eot* – even-odd transition [20];
- *eoti* – even-odd transition iterative [26];
- *eotirt* – even-odd transition iterative [26] to which the real-time data acquisition (see Fig. 8) was applied.

TABLE I. THE VALUES L(N), η(N), AND C(N) FOR DIFFERENT DATA SORTERS

	N=64	**N=128**	**N=256**	**N=512**	**N=1,024**
$L_2(N)$	6	7	8	9	10
$\eta_2(N)$	Combined with data transfers through I/O ports				
$L_8(N)$	1	1	1	1	1
$\eta_8(N)$	≤64	≤128	≤256	≤512	≤1,024
$L_9(N)$	1	1	1	1	1
$\eta_9(N)$	Combined with data transfers through I/O ports				
$L_{eom}(N)$	21	28	36	45	55
$L_b(N)$	21	28	36	45	55
$L_{eot}(N)$	64	128	256	512	1,024
$L_{eoti}(N)$	2	2	2	2	2
$\eta_{eoti}(N)$	≤32	≤64	≤128	≤256	≤512
$L_{eotirt}(N)$	2	2	2	2	2
$\eta_{eotirt}(N)$	Combined with data transfers through I/O ports				
$C_2(N)$	63	127	255	511	1,023
$C_8(N)$	63	127	255	511	1,023
$C_9(N)$	63	127	255	511	1,023
$C_{eom}(N)$	543	1,471	3,839	9,727	24,063
$C_b(N)$	672	1,792	4,608	11,520	28,160
$C_{eot}(N)$	2,016	8,128	32,640	130,816	523,776
$C_{eoti}(N)$	63	127	255	511	1,023
$C_{eotirt}(N)$	63	127	255	511	1,023

Analysis of Table I permits the following conclusions to be drawn:

- The number of combinational levels in the proposed sorters is the smallest enabling higher clock frequency to be attained comparing to the best known alternatives;
- Combining sorting with data transfers permits the sorting time to be either entirely avoided or significantly reduced;
- Resources occupied by the proposed sorters are considerably smaller than those for the best known circuits. This permits larger networks to be built within the same hardware resources.

Note that the values F_{max} and τ can be found only experimentally because it is difficult to take into account

delays in physically implemented circuits which also include signal propagations in connections.

Fig. 10 demonstrates the organization of experiments for which the following two prototyping boards were chosen:

1. Digilent Nexys-4 [28] with Xilinx Artix-7 FPGA xc7a100. This board permits autonomous data sorters to be evaluated easily.
2. Avnet ZedBoard [29] with Xilinx Zynq APSoC xc7z020. This board permits data sorters partially implemented in software (in the PS) and partially in hardware (in the PL) to be verified and evaluated.

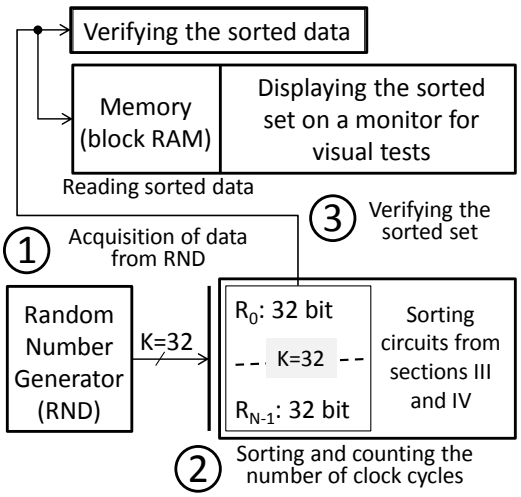

Figure 10. Experimental setup

Initial (unsorted) data are generated randomly and supplied to the proposed circuits (in all experiments the value K=32 was chosen). Sorting is done in networks described in sections III and IV. The results (the sorted set) are verified in FPGA/PL and displayed on a monitor screen for possible visual tests. All the projects were described in VHDL. Synthesis and implementation were done in Xilinx Vivado 2015.2 design suite. The FPGA/PL clock frequency was set to 100 MHz (because this frequency is defined for on-board oscillators [28-29]).

Fig. 11 shows the occupied hardware resources for Nexys-4 board [28] (the number of flip-flops – FF and look-up tables – LUT) taken from Vivado 2015.2 post-implementation reports. The resources are indicated for different circuits namely for Fig. 2, 8, 9 and iterative even-odd transition networks from [26] operating in real-time mode - *eotirt*. We found that the resources of communication-time circuits are smaller than in [26]. The number N was chosen to be 256, 512, and 1,024 for analyzing the largest circuits than can be implemented in the FPGA [28]. Note that for two designs (in Fig. 2 and *eotirt* from [26]) the number of LUTs exceeds the available in FPGA resources and the results for such designs were taken from Vivado synthesis reports.

It is easily visible from Fig. 11 that the circuit in Fig. 9 requires the smallest hardware resources. However, it is clearly seen from section III that the circuits in Fig. 2 and 6 are the fastest.

Fig. 12 shows the resources of the fastest circuits in Fig. 2 and 6 and the less resource consuming circuit in Fig. 9 for ZedBoard [29]. Once again the sorter in Fig. 9 for N=1,024 occupies 86% of APSoC resources while the other sorters (Fig. 2, 6) cannot be implemented.

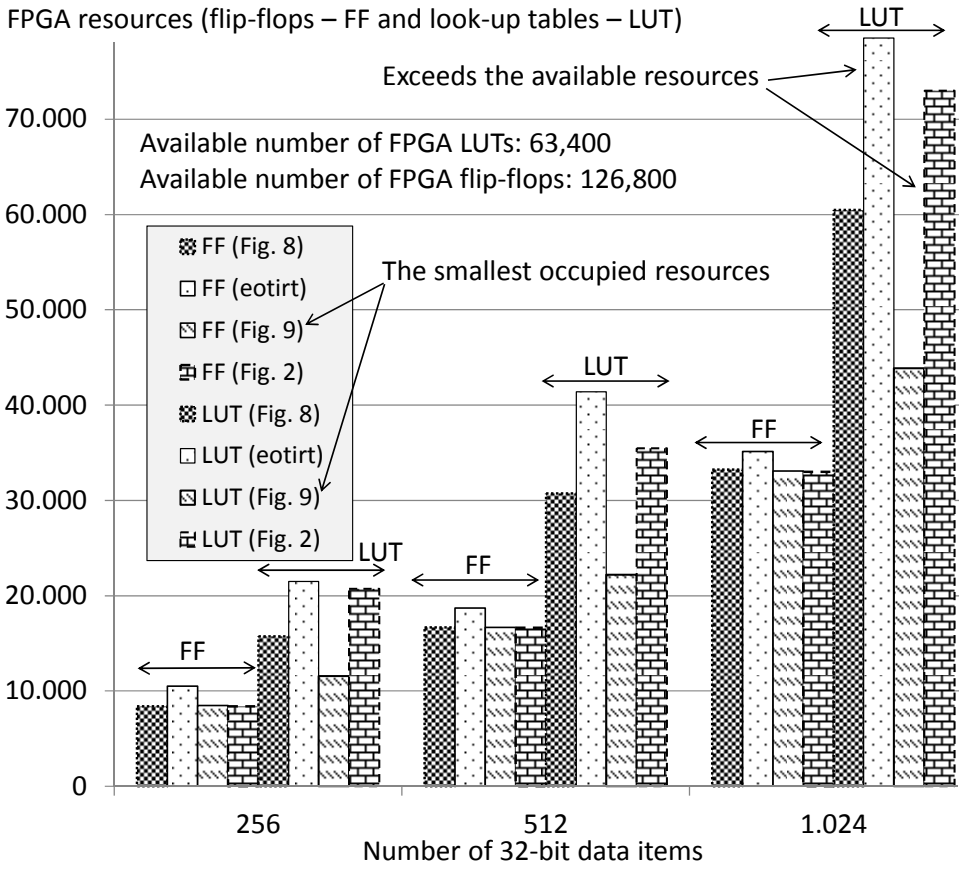

Figure 11. The occupied FPGA resources in Nexys-4 for circuits in Fig. 2, 8, 9 and [26]

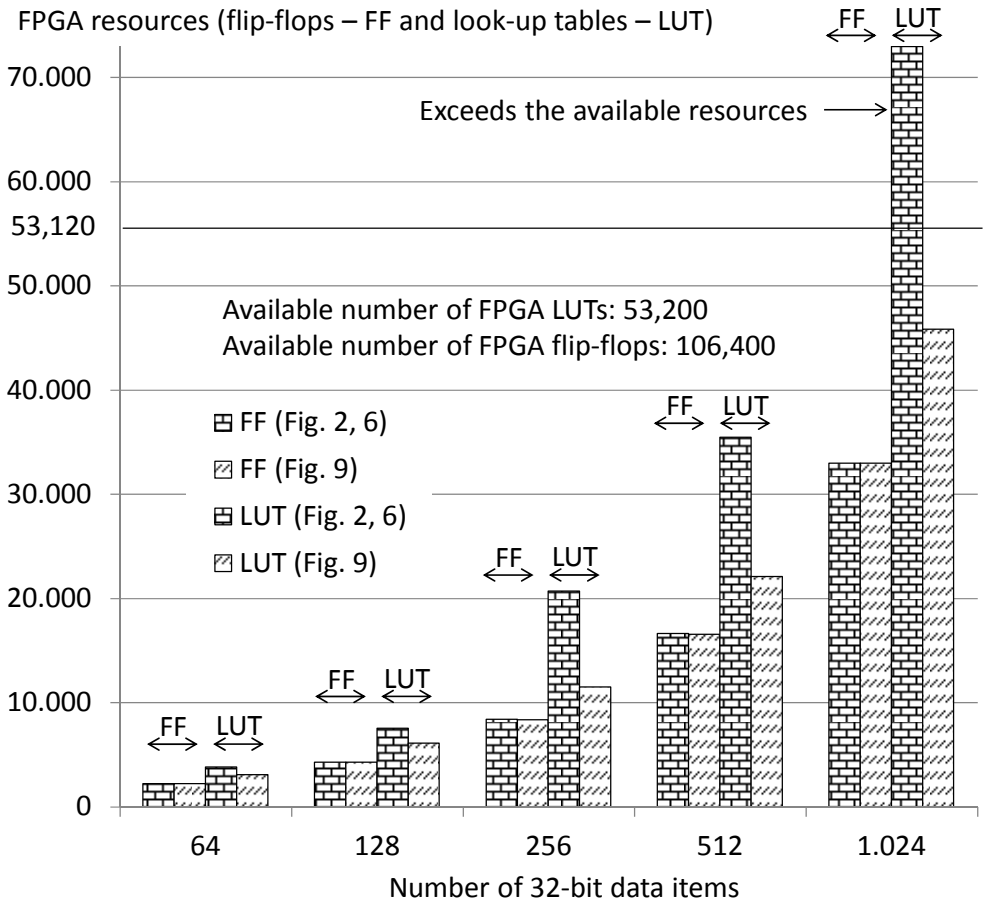

Figure 12. The occupied FPGA resources in ZedBoard for real-time data sorters in Fig. 2, 6, and 9

Table II indicates average number of additional clock cycles (from 100 runs over randomly generated unsorted data items) to produce the sorted set after data acquisition (receiving items from ports) has completed.

TABLE II. AVERAGE NUMBER OF ADDITIONAL CLOCK CYCLES FROM 100 RUNS OVER RANDOMLY GENERATED DATA (N/A – DATA ARE NOT AVAILABLE, BECAUSE THE CIRCUIT CANNOT BE IMPLEMENTED DUE TO INSUFFICIENT FPGA/PL RESOURCES)

	N=64	**N=128**	**N=256**	**N=512**	**N=1,024**
$\eta_2(N)$	5	6	7	8	n/a
$\eta_6(N)$	19.9	26.3	36	46.8	n/a
$\eta_9(N)$	51	104	238	454	992
$\eta_{eotirt}(N)$	25	57	107	209	n/a

Let us analyze Table II. For the circuits in Fig. 2 and 6 sorted data items can be transferred immediately through one port. After $\eta_2(N)/\eta_6(N)$ clock cycles, the sorted data can be transmitted through multiple ports but for such purposes additional multiplexers are needed to select segments of the registers $R_0,\ldots,R_{N-1}$ from which data have to be transferred through the ports (see an example in Fig. 13 for 4 output ports).

We tested also the networks from [24-25] and found that:

- Even-odd merge networks can be implemented in Nexys-4 [28] only for up to N=64 (K=32);
- Bitonic merge networks can be implemented in Nexys-4 [28] only for up to N=56 (K=32);
- Even-odd merge networks can be implemented in ZedBoard [29] only for up to N=50 (K=32);
- Bitonic merge networks can be implemented in ZedBoard [29] only for up to N=44 (K=32);

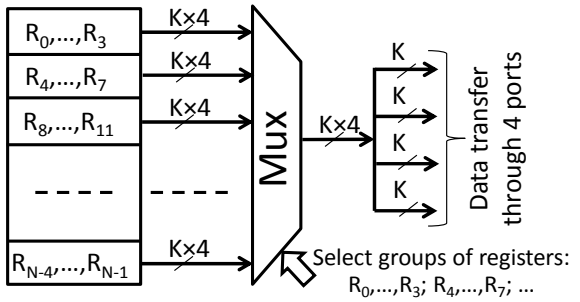

Figure 13. Multiplexing groups of the registers to output ports

Let us compare now the throughput of the proposed circuits with the best known alternatives. Since the known methods [24-25] permit significantly smaller number of data items to be processed in FPGA, more frequent data exchange would be needed which undoubtedly increases communication overhead [10,13]. Besides, merging smaller subsets of data in software requires significantly longer time than merging large subsets. Our experiments with Zynq xc7z020 device have demonstrated significant speed-up of sorting when larger data sets can be processed in programmable logic [13]. Besides, from Table I we can clearly see that the latency of the proposed circuits is significantly smaller than for the best known alternatives (compare values $L_2(N)/L_8(N)/L_9(N)$ and $L_{eom}(N)/L_b(N)/L_{eot}(N)$ in Table I). Thus, the maximum combinational delay in the proposed circuits is smaller and they operate at higher clock frequency than the known circuits. We found that for ZedBoard sorting throughput in

the proposed circuits is much close to the maximum theoretical bandwidth for data transfer through HP AXI and ACP AXI [1]. Non real-time sorting with the aid of the known methods [24-25] is slower because of the following:

- Sorted in hardware blocks have significantly smaller number of items.
- Sorting cannot be done during data transfers and requires additional time.
- Combinational path delays are significantly larger which does not permit high clock frequency to be used.
- Relative communication overheads for transferring smaller blocks are higher than for transferring larger blocks [10,13]. Besides, the fastest burst mode becomes more difficult to be applied for full burst speed.
- Since merging in software is slower than network-based parallel sorting in hardware, beginning the merging with smaller blocks requires significantly more time than beginning the merging with larger blocks.

Experiments were done with hardware/software sorters for Zynq devices from [13] replacing hardware data sorters from [26] by the proposed sorters. Large sets (up to 38 million of 32-bit items) were sorted.

VI. Conclusion

Several architectures of hardware accelerators for data sort are proposed and analyzed. They differ from the known alternatives in two aspects: 1) sorting is done in real-time with transferring input/output data enabling throughput in hardware to be increased; 2) the circuits are very regular and many of them do not require supplementary components such as that are often needed in known designs to form long size vectors from inputs received through limited size ports. The results of experiments and comparisons demonstrated that the proposed circuits are faster and less resource consuming. The results can be used in hardware software co-design and for autonomous high-speed low-cost data sorters.

References

[1] Xilinx, Inc., Zynq-7000 All Programmable SoC Technical Reference Manual, 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[2] L.H. Crockett, R.A. Elliot, M.A. Enderwitz, and R.W. Stewart, The Zynq Book, University of Strathclyde, 2014.

[3] L. Hao and G. Stitt, "Bandwidth-Sensitivity-Aware Arbitration for FPGAs," IEEE Embedded Systems Letters, vol. 4, no. 3, 2012, pp. 73-76. doi: 10.1109/LES.2012.2209397

[4] D.G. Bailey, Design for Embedded Image Processing on FPGAs, John Wiley and Sons, 2011. doi: 10.1002/9780470828519

[5] V. Sklyarov, I. Skliarova, A. Barkalov, and L. Titarenko, Synthesis and Optimization of FPGA-based Systems, Springer, 2014. doi: 10.1007/978-3-319-04708-9

[6] A. Cristo, K. Fisher, A.J. Gualtieri, R.M. Pérez, and P. Martínez, "Optimization of Processor-to-Hardware Module Communications on Spaceborne Hybrid FPGA-based Architectures," IEEE Embedded Systems Letters, vol. 5, no. 4, 2013, pp. 77-80. doi: 10.1109/LES.2013.2286812

[7] A. Canedo, H. Ludwig, and M.A. Al Faruque, "High Communication Throughput and Low Scan Cycle Time with Multi/Many-Core Programmable Logic Controllers," IEEE Embedded Systems Letters, vol. 6, no. 2, 2014, pp. 21-24. doi: 10.1109/LES.2014.2299731

[8] M. Santarini, "All Eyes on Zynq SoC for Smart Vision," XCell Journal, issue 83, 2013, pp. 8-15. [Online]. Available: http://www.xilinx.com/publications/archives/xcell/Xcell83.pdf

[9] C. Dick, "Xilinx All Programmable Devices Enable Smarter Wireless Networks," XCell Journal, issue 83, 2013, pp. 16-23. [Online]. Available: http://www.xilinx.com/publications/archives/xcell/Xcell83.pdf

[10] 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, 2015, pp. 31-34. doi: 10.1109/LES.2015.2399656

[11] Xilinx, Inc., Vivado Design Suite Guides, 2015. [Online]. Available: www.xilinx.com

[12] Xilinx, Inc., Zynq-7000 All Programmable SoC Software Developers Guide, 2015. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf

[13] 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.

[14] Xilinx, Inc., Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors, 2013. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1078-amp-linux-bare-metal.pdf

[15] D.E. Knuth, The Art of Computer Programming. Sorting and Searching, vol. III, Addison-Wesley, New York 2011.

[16] R. Mueller, J. Teubner, and G. Alonso, "Sorting Networks on FPGAs," The International Journal on Very Large Data Bases, vol. 21, no. 1, 2012, pp. 1-23. doi: 10.1007%2Fs00778-011-0232-z

[17] M. Zuluada, P. Milder, and M. Puschel, "Computer Generation of Streaming Sorting Networks," in Proc. 49th Design Automation Conference, San Francisco, USA, 2012, pp. 1245-1253. doi: 10.1145/2228360.2228588

[18] R.D. Chamberlain and N. Ganesan, "Sorting on Architecturally Diverse Computer Systems," in Proc. 3rd Int. Workshop on High-Performance Reconfigurable Computing Technology and Applications, USA, 2009, pp. 39-46. doi: 10.1145/1646461.1646466

[19] D. Koch and J. Torresen, "FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting", in Proc. 19th ACM/SIGDA Int. Symposium on Field Programmable Gate Arrays, USA, 2011, pp. 45-54. doi: 10.1145/1950413.1950427

[20] P. Kipfer, and R. Westermann, "Improved GPU Sorting," in GPU Gems 2: programming techniques for high-performance graphics and general-purpose computation, M. Pharr (ed.), 2005. [Online]. Available: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter46.html

[21] G. Gapannini, F. Silvestri, and R. Baraglia, "Sorting on GPU for large scale datasets: A thorough comparison," Information Processing and Management, vol. 48, no. 5, 2012, pp. 903–917. doi: 10.1016/j.ipm.2010.11.010

[22] C. Grozea, Z. Bankovic, and P. Laskov, "FPGA vs. Multi-Core CPUs vs. GPUs: Hands-On Experience with a Sorting Application," in Facing the Multicore-Challenge, R. Keller, D. Kramer, and J.P. Weiss (eds.), Springer-Verlag, 2010, pp. 105-117. doi: 10.1007/978-3-642-16233-6

[23] M. Edahiro, "Parallelizing fundamental algorithms such as sorting on multi-core processors for EDA acceleration," in Proc. 18th Asia and South Pacific Design Automation Conf., Japan, 2009, pp. 230-233. doi: 10.1109/ASPDAC.2009.4796485

[24] K.E. Batcher, "Sorting networks and their applications," in Proc. American Federation of Information Processing Societies (AFIPS) Spring Joint Computer Conf., USA, 1968, pp. 307-314. doi: 10.1145/1468075.1468121

[25] S.W. Al-Haj Baddar and K.E. Batcher, Designing Sorting Networks. A New Paradigm, Springer, 2011. doi: 10.1007/978-1-4614-1851-1

[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, 2014, pp. 470-484. doi: 10.1016/j.micpro.2014.03.003

[27] V. Sklyarov and I. Skliarova, "Fast regular circuits for network-based parallel data processing," Advances in Electrical and Computer Engineering, vol. 13, no. 4, 2013, pp. 47–50. doi: 10.4316/AECE.2013.04008

[28] Digilent Inc., Nexys4™ FPGA board reference manual, 2013. [Online]. Available: http://www.digilentinc.com/Data/Products/NEXYS4/Nexys4_RM_VB1_Final_3.pdf

[29] Avnet, Inc., ZedBoard Hardware User's Guide, 2014. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf