Research Article

On-Chip Reconfigurable Hardware Accelerators for Popcount Computations

Valery Sklyarov, Iouliia Skliarova, and João Silva

Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, 3810-193 Aveiro, Portugal

Correspondence should be addressed to Iouliia Skliarova; iouliia@ua.pt

Received 26 November 2015; Accepted 21 February 2016

Academic Editor: Eduardo Marques

Copyright © 2016 Valery Sklyarov et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Popcount computations are widely used in such areas as combinatorial search, data processing, statistical analysis, and bio- and chemical informatics. In many practical problems the size of initial data is very large and increase in throughput is important. The paper suggests two types of hardware accelerators that are (1) designed in FPGAs and (2) implemented in Zynq-7000 all programmable systems-on-chip with partitioning of algorithms that use popcounts between software of ARM Cortex-A9 processing system and advanced programmable logic. A three-level system architecture that includes a general-purpose computer, the problem-specific ARM, and reconfigurable hardware is then proposed. The results of experiments and comparisons with existing benchmarks demonstrate that although throughput of popcount computations is increased in FPGA-based designs interacting with general-purpose computers, communication overheads (in experiments with PCI express) are significant and actual advantages can be gained if not only popcount but also other types of relevant computations are implemented in hardware. The comparison of software/hardware designs for Zynq-7000 all programmable systems-on-chip with pure software implementations in the same Zynq-7000 devices demonstrates increase in performance by a factor ranging from 5 to 19 (taking into account all the involved communication overheads between the programmable logic and the processing systems).

1. Introduction

Popcount (which is a short for “population count,” also called Hamming weight) $P(A)$ of a binary vector A is the number of ones in the vector $A = \{a_0, \dots, a_{N-1}\}$. It is also defined for any vector (not obligatory binary) as the number of the vector's nonzero elements. In many practical applications, the execution time for popcount computations over vectors has a significant impact on overall performance of systems that use the results of such computations. They are widely requested in different areas and we will show below just a few examples.

Let us consider the covering problem which can be formulated on sets [1, 2] or matrices [1]. Let $\Theta = (\theta_{ij})$ be a 0-1 incidence matrix. The subset $\Theta_i = \{j \mid \theta_{ij} = 1\}$ contains all columns covered by a row i (i.e., the row i has the value 1 in all columns of the subset Θ_i). The minimal row cover is composed of the minimal number of the subsets Θ_i that cover all the matrix columns. Clearly, for such subsets there is at least one value 1 in each column of the matrix. Different algorithms have been proposed to solve

the covering problem, such as greedy heuristics [1, 2] and a very similar method [3]. Highly parallel algorithms are described in [4]. It is suggested that the given matrix (set) be unrolled in such a way that all its rows and columns are saved in the FPGA registers. Note that more than a hundred of thousands of such registers are available in recent even low-cost devices. This technique permits all rows and columns to be accessed and processed concurrently counting HW for all the rows and columns in parallel.

In recent years genetic data analysis has become a very important research area and the size of data to be processed has been increased significantly. For example, to represent genotypes of 1000 individuals 37 GB array is created [5]. To store such large arrays a huge memory is required. The compression of genotype data can be done in succinct structures [6] with further analysis in such applications as BOOST [7] and BiForce [8]. Succeeding advances in the use of succinct data structures for genomic encoding are provided in [5]. The methods proposed in [5] intensively compute popcounts for very large data sets and it is underlined that

further performance increase can be possible in hardware accelerators of popcount algorithms. Similar problems arise in numerous bioinformatics applications such as [5–12]. For instance, in [9], Hamming distance filter for oligonucleotide probe candidate generation is built to select candidates below the given threshold. The Hamming distance $d(A, B)$ between two vectors A and B is the number of positions they differ in. Since $d(A, B) = P(A \text{ XOR } B)$, the distance can easily be found.

Similarity search is widely used in chemical informatics to predict and optimize properties of existing compounds [13, 14]. A fundamental problem is to find all the molecules whose fingerprints have Tanimoto similarity no less than a given value. It is shown in [15] that solving this problem can be transformed to Hamming distance query. Many processing cores have the relevant instructions; for instance, POPCNT (population count) [16] and VCNT (Vector Count Set Bits) [17] are available for Intel and ARM Cortex microchips, respectively. Operations (like POPCNT and VCNT) are needed in numerous applications and can be applied to very large sets of data (see, e.g., [14]).

Popcount computations are widely requested in many other areas. Let us give a few examples. To recognize identical web pages, Google uses SimHash to get a 64-dimension vector for each web page. Two web pages are considered as near-duplicate if their vectors are within Hamming distance 3 [18, 19]. Examples of other applications are digital filtering [20], matrix analyzers [21], piecewise multivariate functions [22], pattern matching/recognition [23, 24], cryptography (finding the matching records) [25], and many others.

The paper proves that popcount computations can be done in FPGA significantly faster than in software. The following new contributions are provided:

- (i) Highly parallel methods in FPGA-based systems which are faster than existing alternatives.
- (ii) A hardware/software codesign technique implemented and tested in recent all programmable systems-on-chip from the Xilinx Zynq-7000 family.
- (iii) Data exchange between software and hardware modules through high-performance interfaces in such a way that the implemented burst mode enables run-time popcounts computations to be combined with data transfer avoiding any additional delay.
- (iv) The result of experiments and comparisons demonstrating increase of throughput comparing to the best known hardware and software alternatives.

The remainder of the paper contains 6 sections. Section 2 presents a brief overview of related work and analyzes highly parallel circuits for popcount computations. Section 3 suggests system architectures for the two proposed design techniques. Particular solutions with experiments and comparisons are presented in Sections 4 and 5. Section 4 is dedicated to FPGA-based designs and Section 5 to the designs based on all programmable systems-on-chip from the Xilinx Zynq-7000 family. Section 6 discusses the results. Section 7 concludes the paper.

2. Related Work

State-of-the-art hardware implementations of popcount computations have been exhaustively analyzed in [26–30]. The results were presented in form of charts in [26, 28, 30] that compare the cost and the latency of four selected methods. The basic ideas of these methods are summarized below:

- (1) Parallel counters from [26] are tree-based circuits that are built from full-adders.
- (2) The designs from [27] are based on sorting networks, which have known limitations; in particular, when the number of source data items grows, the occupied resources are increased considerably.
- (3) Counting networks [28] eliminate propagation delays in carry chains that appear in [26] and give very good results especially for pipelined implementations. However, they occupy many general-purpose logical slices which are very extensively employed for the majority of practical applications frequently running parallel with popcount computations.
- (4) The designs [30] are based on embedded to FPGA digital signal processing (DSP) slices that either use a very small number of logical slices or do not use them at all.

Different software implementations in general-purpose computers and application-specific processors are also very broadly discussed [14, 29, 31]. A number of benchmarks are given in [14] which will be later used for comparisons. Since hardware circuits allow high-level parallelism to be provided they are faster and we will prove it in Sections 4–5. Besides, popcount computations for long vectors, required for a number of applications, involve multiple data exchange with memory that can be avoided in FPGA-based solutions where the implemented circuits can easily be customized for any size of vectors.

We suggest here novel designs for popcount computations giving better performance than the best known alternatives. All the results will thoroughly be evaluated and compared with existing solutions on available benchmarks (such as [14]).

FPGAs operate on a lower clock frequency than nonconfigurable application-specific integrated circuits and broad parallelism is evidently required to compete with potential alternatives. Let us use such circuits that enable to process in parallel as many bits of a given binary vector as possible.

One feasible approach is based on the frequently researched networks for sorting [27, 31]. However, they are very resource consuming [32]. In [28] a similar technique was used for parallel vector processing with noncomparison operations. The proposed circuits are targeted mainly towards various counting operations and they are called counting networks. In contrast to competitive designs based on parallel counters [26], counting networks do not involve a carry propagation chain needed for adders in [26]. Thus, the delays are reduced and this is clearly shown in [28]. The networks [28] are easily parameterizable and scalable allowing thousands of bits to be processed in combinational circuits. Besides,

a pipeline can easily be created. A competitive circuit can be built directly from FPGA look-up tables (LUTs) using the methods [33]. A LUT(n, m) with n inputs and m outputs can be configured to implement arbitrary Boolean functions $f_0, \dots, f_{m-1}$ of n variables $x_0, \dots, x_{n-1}$. In recent FPGAs (e.g., the Xilinx 7th series and the Altera Stratix V family), most often n is 6 and m is either 1 or 2. If we consider the FPGA generations during the last decade, we can see that these values (n , in particular) have been periodically increased. Clearly, h elements LUT(n, m) can be configured to calculate the popcount $P(A)$ of $A = \{a_0, \dots, a_{n-1}\}$, where the number of LUTs $h = \lceil (\log_2(n+1))/m \rceil$. It is important to note that the delay is very small (e.g., in the Xilinx 7th family FPGAs it is less than 1 ns). The idea is to build a network from LUTs(n, m) that can calculate the popcount for an arbitrary vector A of size N . For filtering problems that appear, in particular, in genetic data analysis this weight is compared with either a fixed threshold κ or the popcount $P(B)$ of another binary vector B to be found similarly.

From experiments in [28, 30, 33] we can see that counting networks and LUT- and DSP-based circuits are the fastest comparing to other alternative methods and we will base popcount computations on a combination of them.

3. System Architectures

Data that have to be processed are kept in memories with capacity of up to tens of GB [4, 5]. Thus, we need to transmit very large volumes of data to the counter (computing popcounts) and this process involves communication time that can exceed the processing time. We suggest the following two design techniques targeted to FPGA and to all programmable systems-on-chip (APSoCs) [34]:

- (1) FPGA-based accelerators for general-purpose computers with architecture shown in Figure 1. The complexity of recent FPGAs permits the complete system (or large subsystem of the system) to be entirely implemented in hardware and accelerators (like those computing popcounts) are the system components.
- (2) APSoC responsible for solving a relatively independent problem and potentially interacting with a general-purpose computer as it is shown in Figure 2.

The first design (see Figure 1) contains an FPGA-based system that solves either a complete problem (such as exemplified in Section 1) or is dedicated to subproblems involving popcount computations. In the last case, the FPGA is used as a hardware accelerator for general-purpose software running in the host PC. Since the paper is dedicated to popcount computations, only one block from Figure 1 (pointed to by the arrow $\swarrow$) will be analyzed. Large input vectors are built inside the FPGA and they are saved either in internal registers or in built-in block RAM. Note that even low-cost FPGAs (such as Artix-7 xc7a100t-1csg324c available on the Nexys-4 prototyping board [35]) contain more than 100 thousands flip-flops and the most advanced FPGAs include millions of flip-flops. Available 140 36 Kb Block RAMs in the FPGA of the Nexys-4 board [35] can be configured to be up to 72 bits in

FIGURE 1: FPGA-based accelerator for a general-purpose computer.

FIGURE 2: APSoC-based accelerator for a general-purpose computer.

width and thus $72 \times 140 = 10,080$ bits can be read or written in parallel. More advanced FPGAs possess almost 2000 of such blocks.

The second design (see Figure 2) contains an APSoC-based accelerator that either solves complete problems indicated in Section 1 or is dedicated to subproblems. We target our designs to Zynq-7000 family of APSoCs [34] which embed a dual-core ARM® Cortex™-A9 MPCore™-based processing system (PS) and the Xilinx 7th family programmable logic (PL) from either Artix-7 or Kintex-7 FPGAs.

FIGURE 3: Interactions between the basic functional components in the Zynq-7000 APSoC.

In contrast to Figure 1 we will discuss a three-level processing system including the following components [36]:

- (1) A general-purpose computer (such as PC) running application-specific software.
- (2) The PS running application-specific software.
- (3) The PL implementing application-specific hardware.

On-chip interactions between the PS and PL are shown in Figure 3 (additional details can be found in [34]).

There are 9 Advanced eXtensible Interface (AXI) ports between the PS and PL that involve over a thousand of on-chip signals [34]. Large size vectors for popcount computations will be received by the PL from memories (double data rate, DDR, on-chip memory, OCM, or cache) through up to 5 AXI ports that are as follows:

- (i) One 64-bit accelerator coherency port (ACP) indicated by letter A in Figure 3 which allows to get data from the ARM cache, OCM, or external DDR memory.
- (ii) Four 32/64-bit high-performance (HP) ports (marked with letter B in Figure 3) allowing to get data from either external DDR memory or OCM.

According to [34], the theoretical bandwidth for read operations through any port listed above is 1200 MB/s (in case of OCM it is 1779 MB/s) and we will evaluate the actual performance for the chosen APSoC later on in the paper.

The resulting popcount will be sent to the PS through one 32-bit general-purpose (GP) port indicated by letter C in

Figure 3. One 32-bit port enables popcounts for $N = 2^{32} - 1$ to be transmitted in one transaction. Since the theoretical bandwidth is 600 MB/s [34] we can neglect the relevant delay. Popcounts will be computed in the PL using logical slices, block RAM, and DSP slices. A combination of the methods [28, 30, 33] will be used and acceleration comparing to software will be measured and reported.

Data exchange between the APSoC and a host PC (see Figures 1 and 2) is not the main target of the paper and it can be organized through a high-performance PCI express bus or USB. In the experiments below data for analysis are created in the host PC and supplied to the FPGA/APSoC through the following:

- (1) On-chip memories using projects from [36].
- (2) Files copied to large DDR memory (see Figure 3) using projects from [36].
- (3) PCI express (in projects with the FPGA available on the VC707 prototyping board [37] that are based on Xilinx IP cores).

4. Design and Evaluation of FPGA-Based Accelerators

Figure 4 depicts the evaluated architecture for popcount computations in an FPGA-based accelerator.

We found that the fastest result could be obtained in a composition of pre- and postprocessing blocks because of the following reasons. It is shown that LUT-based circuits [33] and counting networks [28] are the fastest solutions

FIGURE 4: The proposed architecture for popcount computations in an FPGA-based accelerator.

comparing to the existing alternatives for small subvectors with such sizes η that are 32, 64, or 128 bits. For example, the designs from [33] enable popcounts for $N = 32$ to be found in about 3.5 ns (in the low-cost FPGA xc7a100t-1csg324c available on the Nexys-4 board [35]). Similar computations can be organized as a tree of DSP adders [30]. To compute popcounts for $N = 32$, five sequential DSP adder tree levels are needed [30] involving five DSP delays that are greater than the delays for networks [33].

The resources occupied by the networks from [33] are insignificant for small values of η (such as 32, 64, or 128) and they are rapidly increased for larger values of η . We will show below that DSP-based circuits [30] are more economical for postprocessing. Numerous experiments have demonstrated that a compromise between the number of logical and DSP slices can be found dependently on the following:

- (1) Utilization of logical/DSP slices for other circuits implemented on the same FPGA (i.e., the unneeded for other circuits FPGA resources can be employed for popcount computations).
- (2) Optimal use of available resources in such a way that allows the largest vectors to be processed in a chosen microchip. For example, we found that for the xc7a100t-1csg324c FPGA available on the Nexys-4 board [35] the largest vector (with the size exceeding 40,000 bits) can be handled for $\eta = 32$. For a Virtex-7 FPGA available on the board [37] hundreds of thousands of bits can be handled concurrently.

Let us consider an example for $N = 256$ and $\eta = 32$ that is shown in Figure 5.

FIGURE 5: An example of popcount computations for $N = 256$ and $\eta = 32$.

Single instruction, multiple data (SIMD) feature allows the 48-bit logic unit in the DSP slice [38] to be split into four smaller 12-bit segments (with carry out signal per segment) performing the same function. The internal carry propagation between the segments is blocked to ensure independent operations. The described above feature enables only two DSP slices to be used (from 240 DSP slices available in the low-cost FPGA xc7a100t-1csg324c [35]) and preprocessing is done with only 112 logical slices (from 15,850 logical slices available). Similarly, more complicated designs for popcount computations can be developed.

We synthesized, implemented, and tested circuits for popcounts and compared them with benchmarks from [14] where general-purpose computers with multicore processors were used for similar computations in software. Table 1 presents the results of synthesis, implementation, and test, where N is the size of vectors in bits, N_{DSP} is the number of the occupied DSP slices, N_s is the number of the required logical slices, LUTs is the number of LUTs, FFs is the number of flip-flops, and L is the number of levels in the DSP-based tree from adders. The percentage of the used resources is also shown near the relevant numbers. Please note that the percentage was calculated for different microchips which have different available resources. The clock frequency was set to only 50 MHz. All the design steps were done in Xilinx Vivado 2014.4. The number of slices was calculated as the number of LUTs from Vivado reports divided by 4. Experiments were done for three different prototyping boards that are explicitly indicated (Nexys, Nexys-4 [35], Zed, ZedBoard [39], and Zy,

TABLE 1: The results of experiments ($\eta = 32$).

N	8,192	10,416	20,832	31,248	41,664
N_{DSP}	44 (55%)	55 (69%)	109 (50%)	162 (74%)	215 (90%)
N_s	3,438	4,056	7,920	10,859	14,266
LUTs	13,752 (78%)	16,221 (92%)	31,679 (60%)	43,434 (82%)	57,063 (90%)
FFs	8,439 (24%)	10,597 (30%)	21,158 (20%)	21,158 (20%)	43,211 (34%)
L	8	9	10	10	11
Board	Zy	Zy	Zed	Zed	Nexys

ZyBo [40]). Clearly, the board [37] permits significantly more complicated designs to be developed.

To reduce the delay, output registers in the DSP48E1 slices [38] are synchronized by clock and the result is computed in L clocks cycles. Thus, the delay from getting an N -bit vector on the circuit inputs to producing the result is $20 \times L$ ns (we remind that clock frequency is set to 50 MHz and the clock period is 20 ns). It means that popcounts are computed as fast as from 160 ns (for $L = 8$) to 220 ns (for $L = 11$).

Let us compare the results with [14], where the fastest popcounts for 8 MB vectors are computed in 242,884 μ s. Thus, for sizes N in Table 1, our popcount computations are faster by a factor ranging from 185 to 685 (provided the source data are available in FPGA built-in memory). Note that such acceleration can be achievable only in FPGAs with larger built-in memories that have to be at least 8 MB; otherwise communication overheads with external memories need to be taken into account. To process large vectors (such as that are taken for the experiments in [14]) the circuits in Figure 4 need to be reused for vector segments (of size N given in Table 1) with accumulating the results. The latter can also be done in one DSP slice [38]. This gives an additional delay (20 ns for our case). Therefore, the acceleration is slightly reduced. However, accumulating the results can be done in pipeline (i.e., described in [28, 33]). Thus, the acceleration will in fact be increased because only the first segment will be handled in $D_{\max} \times (L + 1)$ ns (1 is added for the last DSP-based accumulator) and all the subsequent segments will be added to the accumulator in $D_{\max}$ ns, where $D_{\max}$ is the maximum delay of circuits between the pipeline registers (e.g., 20 ns for our example). So, the proposed circuits significantly outperform the functionally equivalent software running in multicore general-purpose processors [14].

Additional experiments were done with the prototyping board VC707 [37] with the advanced Virtex-7 XC7VX485T-2FFG1761C FPGA. The largest circuit from Table 1 occupies 215 DSP slices (from 2800 available slices, i.e., less than 8%), 14,300 logical slices (from 75,900 available slices, i.e., less than 19%), and 43,300 flip-flops (from 607,200 available flip-flops, i.e., less than 8%). Thus, the FPGA has sufficient remaining resources for solving additional problems such as [4]. In real applications, the indicated above theoretical speedup (ranging from 185 to 685) is not attainable when we include communication circuits which decrease the acceleration. However, the complexity of reconfigurable devices is dramatically increased and this tendency will undoubtedly be maintained in the future. Thus, complete systems

implemented in FPGA with embedded high-performance multicore processors can be expected in future. For such future systems the results of the paper relevant to popcount computations in FPGA-only circuits are very helpful and important. They prove experimentally that acceleration can be very significant.

To avoid any confusion we provided real experiments and measured throughputs for the VC707 prototyping board [37] connected to a host PC (i7-4770 CPU 3.4 GHz), working under Linux operating system, through a PCI express. The circuits from Table 1 were taken for the experiments. The throughput for PC-FPGA system was slightly increased comparing to the best multicore program from [14] executed on the same PC. However, the bottleneck is in communication overheads. We found that the throughput could be increased additionally by parallelizing the computations between the PC and FPGA-based circuits. However, it is still relatively small. The main conclusion is the following. An FPGA has to implement additional circuits that need the results of popcount computations. For example, using the methods [4], a large binary matrix is transferred and the FPGA executes a very large number of popcount computations for the rows/columns of the matrix to find out either the maximum or the minimum values. Then some rows and columns are masked and the same operations are applied to the remaining part of the matrix. Thus, large volume data need to be transferred through the PCI only ones and then are handled repeatedly gaining advantages of high-performance parallel computations in hardware.

5. Design and Evaluation of APSoC-Based Accelerators

We describe here the following two types of popcount computations in the Xilinx Zynq-7000 APSoCs:

- (1) Software programs running in the PS (i.e., in the Cortex-A9 MPCore).
- (2) Software/hardware codesign where popcounts are computed in hardware and used in software.

The maximum clock frequency for ARM (the PS) in different Zynq-7000 APSoCs microchips ranges from 667 MHz to 1 GHz [34]; we used a microchip with 667 MHz. The clock frequency for the PL was set to 150 MHz. Popcounts for long binary vectors (such as that used in [14]) are computed as follows (see Figure 6):

FIGURE 6: General structure of the project.

- (1) The source vector A is built in the PS and saved in the external DDR memory (512 MB external memory is available on the used prototyping boards [39, 40]).
- (2) On Start signal from the PS, the PL reads segments of the vector (through high-performance ports) for subsequent popcount computations.
- (3) The vector is split into U segments $S_0(A), \dots, S_{U-1}(A)$ with equal number of bits η and popcounts for each segment S_u ($0 \leq u \leq U - 1$) are found as shown in Figure 7. The fastest known circuits (referenced above) are chosen to compute the popcount for η bits and to accumulate popcounts (i.e., to add the currently computed popcount to the sum of all the previously received and computed popcounts for η -bit subvectors). It is important to note that the indicated above operations are executed in parallel with reading η bits in the fastest burst mode; that is, no additional time is required. Control of burst read is done by a dedicated module in a hierarchical finite state machine [41]. This module can be reused in any similar application. We assume that η is equal to the bus size or to the sum of bus sizes if many ports are involved in parallel.
- (4) The final result is produced as a combinational sum of the accumulated popcounts from all the ports (see Figure 7) that can be done either in DSP slices shown in Figure 7 or in a circuit built from logical slices. Any way can be chosen and it depends on availability of either DSP or logical slices and their need for other circuits.

- (5) Since popcounts are incrementally accumulated with the speed of burst transactions
  - (a) millions of bits can easily be processed;
  - (b) there is no faster way because the speed of data transfer in burst mode is predefined by the APSoc characteristics.
- (6) As soon as the popcount is computed, the PL generates an interrupt that forces the PS to read the popcount through a GP port (see Figure 6) and then the popcount can be used for further processing.
- (7) The PS also executes similar operations in software only with the aid of the following functions:
  - (a) A naive function `popcount_software_naive` that sequentially selects bits of the given vector and adds them.
  - (b) The best parallel function from [42].
  - (c) A function `popcount_software_table` that uses look-up tables with 8 entries [42].
  - (d) A function `popcount_software_builtin` that calls the built-in function `_builtin_popcount` [14].
- (8) Finally, the comparison of the best software and hardware results is done in the PS and shown.

Experiments that were carried out with two APSoc-based prototyping boards [39, 40] permit to conclude the following:

- (1) Although the maximum acceleration is achieved with 5 parallel high-performance ports (4 AXI HP ports and 1 AXI ACP port), it is not significant comparing to processing through a smaller number of ports. We think that the bottleneck is in a shared access to the common DDR memory that is used by built-in Zynq-7000 memory controllers.
- (2) We found that the 64-bit AXI ACP port does allow significant additional acceleration comparing to a 32-bit AXI ACP port mainly for $N \leq 64 \times 2^{15}$.

We studied also multicore (dual-core) implementations in the PS and found that they might be advantageous if one core supports hardware computations of popcounts and another core executes the remaining tasks for different types of data analysis (such as that were overviewed in Section 1).

Tables 2–7 present the results that include all the involved communication overheads. The row N_{64} indicates the number of bits in the input vector divided by 64. For example, the column 2^{20} presents the results for popcount computations over 64×2^{20} -bit vectors that are the same as in benchmarks [14]. The row Acc indicates acceleration of hardware popcount computations relatively to the best software popcount computations (implemented in the PS).

Tables 2 and 3 allow communication overheads for the ACP port to be evaluated. The first table uses 32-bit burst transactions (from the available 64 bits). The second table uses full 64-bit burst transactions. As you can see, acceleration is increased up to 64×2^{15} -bit vectors and then

FIGURE 7: Popcount computations in the PL reading data through 4 high-performance ports in burst mode.

TABLE 2: The results of experiments (data are transferred through one 32-bit ACP high-performance port and $\eta = 32$).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	12.11	14.30	6.83	6.04	5.84	5.81

TABLE 3: The results of experiments (data are transferred through one 64-bit ACP high performance port and $\eta = 64$).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	16.95	19.65	7.13	6.16	5.93	5.90

TABLE 4: The results of experiments (data are transmitted through four 32-bit high-performance ports and $\eta = 32$).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	5.56	6.23	6.83	6.96	6.99	7.01

TABLE 5: The results of experiments (data are transmitted through four 64-bit high-performance ports and $\eta = 64$).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	5.69	6.25	7.05	7.29	7.37	7.37

TABLE 6: The results of experiments (data are transmitted through four 32-bit high-performance ports and one 64-bit ACP).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	5.14	6.47	7.08	7.18	7.21	7.21

TABLE 7: The results of experiments (data are transmitted through four 64-bit high-performance ports and one 64-bit ACP).

N_{64}	$2^{10} = 1024$	2^{15}	2^{18}	2^{20}	2^{23}	2^{25}
Acc	5.16	6.66	7.31	7.42	7.44	7.45

decreased. It is easy to explain such results. The ACP port can use cache memory that is fast [34]. As soon as the requested

size of the cache is not available, other memories are used that are slower. There is another interesting feature. The 64-bit ACP port is faster if cache memory is involved; otherwise, the acceleration is negligible comparing to 32-bit mode.

Tables 4 and 5 demonstrate that using four high-performance AXI ports permits better acceleration to be achieved comparing to one AXI ACP port for $N_{64} > 2^{18}$. Although 64-bit AXI ports are faster (than 32-bit ports) accelerations are not very significant.

The fastest popcount computations for $N_{64} \geq 2^{18}$ are done in the hardware/software system with four 64-bit AXI HP ports and one 64-bit AXI ACP port (see Table 7). Using 32-bit AXI ports (see Table 6) is a bit slower but acceleration is also valuable.

The columns marked with 2^{20} permit the results to be compared with benchmarks from [14]. For experiments in Table 6 the computation was done in 7,514,703 units measured by the function XTime_GetTime [43]. Each unit returned by this function corresponds to 2 clock cycles of the PS [43]. The PS clock frequency is 667 MHz. Thus, the clock period is 1.5 ns and $7,514,703 \times 2 = 15,029,406$ clock cycles or 22,544,109 ns = 22,544 μ s are required to produce the result. The fastest program from [14] computes the result for similar data in 242,884 μ s. Thus, the proposed hardware/software popcount computations are faster by a factor of more than 10 and this acceleration includes all the required communication overheads. Note that the comparison is done between a general-purpose computer with a multicore Intel processor running with clock frequency 3.46 GHz [14] and the simplest microchip from the Zynq-7000 family available on ZyBo [40]. Besides, even for such APSoc the used resources are small allowing additional circuits to be accommodated on the same microchip. Table 8 resumes the utilized postimplementation hardware resources from the report in Vivado 2014.4. Only LUTs were chosen. If DSP slices are used for the circuit in Figure 7 then the number of LUTs is reduced.

Two types of popcount computations are used to get the results for Tables 2–7.

FIGURE 8: Popcount computations for using 5 AXI ports in burst mode.

TABLE 8: Postimplementation resources for Table 6 from the Vivado 2014.4 report.

Resource	Utilization	Available	Utilization%
Flip-flops	7,420	35,200	21
Look-up tables (LUTs)	6,398	17,600	36
Memory LUTs	392	6,000	7
BUFG-Xilinx buffers	1	32	3

In the first type, popcounts for all 32-bit ports are computed as shown in Figure 7. Popcounts for 64-bit ports are computed similarly and the only difference is in an additional popcount circuit that was taken from [33] for $\eta = 64$. In the second type, popcounts for five ports are computed as shown in Figure 8.

Accumulating the weights is done in the DSP slice [38]. The values P_1 and P_2 are added and accumulated in one clock cycle which can be done thanks to the ALU with three operands in the DSP48E1 slice.

Note that all the results were obtained in physical tests running in prototyping boards with the aid of methods and Zynq-7000 projects from [36].

6. Discussion of the Results

We described above two architectures and design techniques targeted to FPGAs and Zynq-7000 APSOCs (see Section 3). The first technique is very efficient when the complete system (or large part of the system, such as that is described in [4]) is implemented in FPGA. Acceleration of popcount computations that are used in the system can be very significant. Examples in Section 4 demonstrate speedup by a factor ranging from 185 to 685 comparing to the functionally equivalent software running in multicore general-purpose

computers. However, such an acceleration is considerably reduced if an exchange of large volume data is involved just for popcount computations. This is currently true even for very high-speed interfaces, such as PCI express. Thus, FPGA has to implement more extensive computations that need popcounts (see the final part of Section 4). Advances in high-level synthesis from general-purpose languages (such as C/C++) [44] would undoubtedly permit to simplify the complexity of the design process making it widely acceptable not only by hardware but also by software engineers. Thus, we can expect that systems like [4] can be developed more easily. Some design examples from high-level specifications are given in [36] for Zynq-7000 APSOCs.

The second technique permits a very precise comparison of software and hardware to be done. Although the achieved acceleration is not as significant as for the first technique, all supplementary factors (such as communication overheads and particularities of APSOC controllers) were taken into account and concrete time delays were measured. So, we can talk about the exact comparison that is done for the same microchip. The achieved acceleration in hardware comparing to the best implementations in software ranges from 5.14 to 19.65. Besides, we believe that this acceleration is the maximum possible for particular microchips because no additional time is involved comparing to just data transfer from memory in the fastest burst mode. Thus, to improve the results in hardware it is necessary to provide support for better bandwidth for high-performance ports. Additional acceleration can be achieved if data (that are partially ready) are copied to hardware while other software parts are involved in solving parallel tasks that, in particular, have to prepare the complete set of data for the PL. This problem is outside of the scope of the paper. Besides, popcount computations for very large vectors can be partially implemented in software running in the dual-core PS and partially in hardware in

such a way that the cores and hardware accelerators operate concurrently. This approach permits to reduce the volume of transferred data and additional speedup will undoubtedly be achieved.

7. Conclusion

The main contribution of the presented work is the novel technique for the design of hardware accelerators for popcount computations that are widely used in a broad area of practical applications reviewed in the paper. Two types of highly parallel designs for FPGAs and all programmable systems-on-chips are proposed. The results of experiments with these designs implemented and tested in hardware demonstrate the significant speedup comparing to the functionally equivalent software programs running in multicore processors.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgment

This research was supported by Portuguese National Funds through Foundation for Science and Technology (FCT), in the context of Project UID/CEC/00127/2013.

References

- [1] K. H. Rosen, J. G. Michaels, J. L. Gross, J. W. Grossman, and D. R. Shier, Eds., *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, Boca Raton, Fla, USA, 2000.
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, Mass, USA, 3rd edition, 2009.
- [3] A. Zakrevskij, Y. Pottosin, and L. Cheremisiniva, *Combinatorial Algorithms of Discrete Mathematics*, TUT Press, 2008.
- [4] V. Sklyarov, I. Skliarova, A. Rjabov, and A. Sudnitsin, "Fast matrix covering in all programmable systems-on-chip," *Elektronika ir Elektrotehnika*, vol. 20, no. 5, pp. 150–153, 2014.
- [5] P. P. Putnam, G. Zhang, and P. A. Wilsey, "A comparison study of succinct data structures for use in GWAS," *BMC Bioinformatics*, vol. 14, no. 1, article 369, 2013.
- [6] G. Jacobson, "Space-efficient static trees and graphs," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (SFCS '89)*, pp. 549–554, Research Triangle Park, NC, USA, November 1989.
- [7] X. Wan, C. Yang, Q. Yang et al., "BOOST: a fast approach to detecting gene-gene interactions in genome-wide case-control studies," *The American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, 2010.
- [8] A. Gyenesei, J. Moody, A. Laiho, C. A. M. Semple, C. S. Haley, and W.-H. Wei, "BiForce toolbox: powerful high-throughput computational analysis of gene-gene interactions in genome-wide association studies," *Nucleic Acids Research*, vol. 40, no. 1, pp. W628–W632, 2012.
- [9] C. Hafemeister, R. Krause, and A. Schliep, "Selecting oligonucleotide probes for whole-genome tiling arrays with a cross-hybridization potential," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 6, pp. 1642–1652, 2011.
- [10] O. Milenkovic and N. Kashyap, "On the design of codes for DNA computing," in *Coding and Cryptography*, O. Ytrehus, Ed., vol. 3969 of *Lecture Notes in Computer Science*, pp. 100–119, Springer, Berlin, Germany, 2006.
- [11] A. M. Bolger, M. Lohse, and B. Usadel, "Trimmomatic: a flexible trimmer for Illumina sequence data," *Bioinformatics*, vol. 30, no. 15, pp. 2114–2120, 2014.
- [12] T. D. Wu and S. Nacu, "Fast and SNP-tolerant detection of complex variants and splicing in short reads," *Bioinformatics*, vol. 26, no. 7, pp. 873–881, 2010.
- [13] R. Nasr, R. Vernica, C. Li, and P. Baldi, "Speeding up chemical searches using the inverted index: the convergence of chemoinformatics and text search methods," *Journal of Chemical Information and Modeling*, vol. 52, no. 4, pp. 891–900, 2012.
- [14] Dalke Scientific Software, *Faster Population Counts*, 2011, http://dalkescientific.com/writings/diary/archive/2011/11/02/faster_popcount_update.html.
- [15] X. Zhang, J. Qin, W. Wang, Y. Sun, and J. Lu, "HmSearch: an efficient hamming distance query processing algorithm," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management (SSDBM '13)*, Baltimore, Md, USA, July 2013.
- [16] Intel Corporation, "Intel® SSE4 Programming Reference," 2007, <https://software.intel.com/sites/default/files/m/8/b/8/D9156103.pdf>.
- [17] ARM, *NEON™ Version: 1.0 Programmer's Guide*, 2013, <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0018a/index.html>.
- [18] G. S. Manku, A. Jain, and A. D. Sarma, "Detecting near-duplicates for web crawling," in *Proceedings of the 16th International World Wide Web Conference (WWW '07)*, pp. 141–150, Banff, Canada, May 2007.
- [19] K. Chen, "Bit-serial realizations of a class of nonlinear filters based on positive Boolean functions," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 6, pp. 785–794, 1992.
- [20] P. D. Wendt, E. J. Coyle, and N. C. Gallagher, "Stack filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 898–908, 1986.
- [21] 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.
- [22] M. Storace and T. Poggi, "Digital architectures realizing piecewise-linear multivariate functions: two FPGA implementations," *International Journal of Circuit Theory and Applications*, vol. 39, no. 1, pp. 1–15, 2011.
- [23] K. Asada, S. Kumatsu, and M. Ikeda, "Associative memory with minimum Hamming distance detector and its application to bus data encoding," in *Proceedings of the IEEE Asia-Pacific Application-Specific Integrated Circuits*, pp. 16–18, Seoul, South Korea, 1999.
- [24] C. Barral, J. S. Coron, and D. Naccache, "Externalized fingerprint matching," in *Proceedings of the International Conference on Biometric Authentication (ICBA '04)*, pp. 309–315, Hong Kong, 2004.

- [25] B. Zhang, R. Cheng, and F. Zhang, "Secure Hamming distance based record linkage with malicious adversaries," *Computers and Electrical Engineering*, vol. 40, no. 6, pp. 1906–1916, 2014.
- [26] B. Parhami, "Efficient Hamming weight comparators for binary vectors based on accumulative and up/down parallel counters," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 2, pp. 167–171, 2009.
- [27] S. J. Piestrak, "Efficient Hamming weight comparators of binary vectors," *Electronics Letters*, vol. 43, no. 11, pp. 611–612, 2007.
- [28] V. Sklyarov and I. Skliarova, "Design and implementation of counting networks," *Computing*, vol. 97, no. 6, pp. 557–577, 2015.
- [29] E. El-Qawasmeh, "Beating the popcount," *International Journal of Information Technology*, vol. 9, no. 1, pp. 1–18, 2003.
- [30] V. Sklyarov and I. Skliarova, "Multi-core DSP-based vector set bits counters/comparators," *Journal of Signal Processing Systems*, vol. 80, no. 3, pp. 309–322, 2015.
- [31] D. E. Knuth, *The Art of Computer Programming, Sorting and Searching*, vol. 3, Addison-Wesley, London, UK, 2011.
- [32] 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.
- [33] V. Sklyarov, I. Skliarova, A. Barkalov, and L. Titarenko, *Synthesis and Optimization of FPGA-Based Systems*, Springer, Berlin, Germany, 2014.
- [34] Xilinx, "Zynq-7000 All Programmable SoC Technical Reference Manual," 2015, http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.
- [35] Digilent, *Nexys4™ FPGA Board Reference Manual*, 2013, http://www.digilentinc.com/Data/Products/NEXYS4/Nexys4_RM_VB1_Final_3.pdf.
- [36] 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.
- [37] Xilinx, "VC707 Evaluation Board for the Virtex-7 FPGA User Guide," 2015, http://www.xilinx.com/support/documentation/boards_and_kits/vc707/ug885_VC707_Eval_Bd.pdf.
- [38] Xilinx, *7 Series DSP48E1 Slice User Guide*, Xilinx, San Jose, Calif, USA, 2014, http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.
- [39] Avnet, "ZedBoard (Zynq™ Evaluation and Development) Hardware User's Guide," 2014, http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2.2.pdf.
- [40] Digilent, *ZyBo Reference Manual*, 2014, http://digilentinc.com/Data/Products/ZYBO/ZYBO_RM_B_V6.pdf.
- [41] V. Sklyarov and I. Skliarova, "Hardware implementations of software programs based on hierarchical finite state machine models," *Computers & Electrical Engineering*, vol. 39, no. 7, pp. 2145–2160, 2013.
- [42] S. E. Anderson, "Counting bits set, in parallel," <http://graphics.stanford.edu/~seander/bithacks.html#CountBitsSetParallel>.
- [43] Xilinx, "OS and Libraries Document Collection, Standalone (v.4.1). UG647," 2014, http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_2/oslib_rm.pdf.
- [44] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: from prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.