Hamming Weight Counters and Comparators based on Embedded DSP Blocks for Implementation in FPGA

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

***Abstract*—This paper is dedicated to the design, implementation and evaluation of fast FPGA-based circuits that compute Hamming weights for binary vectors and compare the results with fixed thresholds and variable bounds. It is shown that digital signal processing (DSP) slices that are widely available in contemporary FPGAs may be used efficiently and they frequently provide the fastest and least resource consuming solutions. A thorough analysis and comparison of these with the best known alternatives both in hardware and in software is presented. The results are supported by numerous experiments in recent prototyping boards. A fully synthesizable hardware description language (VHDL) specification for one of the proposed core components is given that is ready to be synthesized, implemented, tested and compared in any FPGA that contains embedded DSP48E1 slices (or alternatively DSP48A1 slices from previous generations). Finally, the results of comparisons are provided that include discussions of designs in an ARM processor combined with reconfigurable logic for very long vectors.**

***Index Terms*—Hamming weight counter, Hamming weight comparator, field-programmable gate array, digital signal processing slice, hardware accelerator, on-chip architecture.**

I. Introduction

The Hamming weight (HW) W(A) of a binary vector A = $\{a_0,\ldots,a_{N-1}\}$ is the number of ones in the vector, which ranges from 0 to N [1]. Certain applications require W(A) to be computed or compared with a fixed threshold k or with W(B), where B = $\{b_0,\ldots,b_{Q-1}\}$ is another binary vector (Q and N can be either equal or different). Examples of such applications are digital filtering [2-4], piecewise multivariate functions [5], pattern matching/recognition [6,7], problem solving in Boolean space [8], combinatorial search [9], encoding for data compression [10], and stream/matrix analyzers [11].

The HW frequently needs to be found in the area of combinatorial search for solving such problems as covering binary matrices [8], Boolean satisfiability [9,12], and graph coloring [13]. The relevant instructions are now embedded to many processing cores. For instance, POPCNT (population count) [14] and VCNT (Vector Count Set Bits) [15] are currently available in Intel and ARM products. Operations (like POPCNT and VCNT) are used very intensively in numerous applications, including those where they are applied to very large data sets (see, for example, [16]). Optimized designs for HW counters targeted to general-purpose processors are proposed (*e.g.* [17]) and broadly discussed, with the main objective being to provide support for significant acceleration.

Many electronic, environmental, medical, and biological applications need to process data that is produced by sensors, and measure external parameters within given upper and lower bounds [11]. In the simplest case there are two thresholds: one for an upper bound and another for a lower bound. Monitoring thermal radiation from volcanic products [18] and digital filtering [4] are examples of such measurements. In more complicated cases, there are several bounds that permit some events to be predicted with higher or lower probability. Dependently on other factors (*e.g.* weather conditions) the bounds may be variable.

It is known that broad parallelism makes significant acceleration possible for a number of practical applications. We will briefly describe one of them. Combinatorial search algorithms (*e.g.* described in [8,9]) frequently require executing operations over binary and ternary matrices, such as finding a row/column with the minimum/maximum HW. Application examples are solving a matrix covering problem [8] and the Boolean satisfiability algorithms [9]. One of the fastest methods is to unroll the matrix and obtain the solution in a combinational circuit that finds HWs for all the unrolled rows/columns and then computes the maximum/minimum value with the aid of the circuits [19]. It should be underlined that the desired HWs can be found in a combinational circuit with just a propagation delay through the logical gates that are used. This is, indeed, very fast. The only problem is that such unrolling cannot be implemented in widely available processing cores without involving sequential computations. However, FPGAs can offer a very elegant solution because the customization that is supported allows the design to be directly mapped to hardware. The size of the matrices can be very large, even for low-cost FPGAs such as the Xilinx Artix-7 that is available on the Digilent Nexys-4 prototyping board [20]. The most important advantages of reconfigurable systems are the inherent configurability and relatively cheap development cost. Recent field-configurable micro-chips incorporate multi-core processors and reconfigurable logic appended with a number of frequently used devices such as digital signal processing (DSP) slices and memories. FPGAs still operate at lower clock frequencies than general-purpose computers and application-specific integrated circuits. The cost of the most advanced devices is high. Cheaper microchips operate at clock frequencies that are lower than

This research was supported by National Funds through FCT - Foundation for Science and Technology, in the context of the project PEst-OE/EEI/UI0127/2014.

Digital Object Identifier 10.4316/AECE.2014.02011

those in inexpensive computers. One of the most important applications of FPGAs is improving the performance of existing systems. To achieve acceleration with devices that are generally slower, parallelism needs to be applied extensively.

The majority of contemporary FPGAs contain embedded DSP slices (*ex.* DSP48E1 slice for Xilinx FPGAs [21]) and they can be employed to implement arithmetical and logical operations. The number of such slices is large. For example, even in the low-cost FPGA Artix-7 [20] there are 240 embedded DSP48E1 slices and in the most advanced FPGAs from the Xilinx 7 series there are more than 3500 such slices. We will show that HW counters and comparators can be built very efficiently on DSPs and this is achievable without the use of general-purpose logical resources. The throughput is high, the cost is reasonable, and it will be demonstrated in the last section that FPGA-based circuits may significantly outperform processor-based implementations. In addition, numerous operations can be executed in parallel. In subsequent sections we will prove that HW counters and comparators on embedded DSP slices are better than known competitors and can be recommended for highly parallel computational systems.

II. Related Work

The state-of-the-art for hardware implementations of HW counters and comparators was analyzed exhaustively in [1,10,11,22,23]. The results were presented in the form of charts that compare the cost (*i.e.* the number of gates) and the latency (*i.e.* the number of gate levels) for five methods [1], [24], [2,25], [22], and [11]. The basic ideas of these methods are summarized below:

1. Parallel counters from [1] are tree-based circuits that are built from full-adders that compute the HW of 3-bit binary sub-vectors (two 1-bit operands and a carry) of the given vector. The 2-bit results are further added and propagated through the tree levels until the final HW is produced. Our comparison has shown that this solution is one of the best.
2. The designs from [24] are based on sorting networks, which have known limitations [11]; in particular, as the number of source data items grows, the resources occupied increase dramatically.
3. Gate-based circuits from [2, 25] are, in fact, bubble-sort networks that are resource consuming [26] and also have long propagation delays. Thus, solutions [24] are generally better when the fastest-known even-odd merge and bitonic merge networks are used [22,26].
4. Counting networks [22] eliminate propagation delays in carry chains in [1] 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. We show below that similar operations can be executed without such slices. Instead, DSP slices are employed which are frequently unused, even though they are available in FPGAs.
5. LUT-based solutions [11] are very economical but they have the same drawbacks as the counting networks.

We suggest novel designs for HW counters and comparators here that provide better performance and consume fewer resources than the best known alternatives [1,2,11,22-25]. The proposed designs also outperform processor-based implementations, which can be verified on benchmarks that are given in [16]. A comparison with software products involving operations like POPCNT [14] and VCNT [15] also demonstrates that the proposed solutions are faster.

III. Embedded DSP-based Hamming Weight Counters and Comparators

The main idea of the method proposed here is the use of a DSP arithmetic and logic unit (ALU) in a way that permits a tree of adders to be built in sections of the same ALU. The sum of two n-bit operands cannot contain more than n+1 bits so the size of each segment is known in advance. Thus for a 16-bit binary vector that is initially represented as eight 2-bit sub-vectors (*i.e.* 8 bits of the first DSP operand and 8 bits of the second DSP operand), the ALU can be segmented as shown in Fig. 1. Initially, the two bits in each of the eight pairs are added. The result of each sum is a maximum of 2 bits, with possible values 00 (*i.e.* 0+0), 01 (*i.e.* 0+1 or 1+0), and 10 (*i.e.* 1+1). The addition is done in eight 2-bit ALU segments, as shown in Fig.1 and Fig. 2a.

The eight 2-bit sums that are produced are further processed as four 4-bit sub-vectors (see Fig. 1). Each sub-vector represents two 2-bit operands that are added to produce a result that is a maximum of 3 bits. These are the inputs of four 3-bit segments of the same ALU (see Fig. 1 and Fig. 2b). In the next stage, two 4-bit segments are allocated (see Fig. 1 and 2c). The last 5-bit segment (see Fig. 1 and 2d) produces the final HW. The outputs of a 48-bit adder [21] for any stage (1-4 in Fig. 1) are considered to be inputs for the next stage of the same DSP, specifying the number of bits that are needed for operands and results at each stage. Similar circuits can also be built in less advanced FPGAs with embedded DSP48A1 slices (*e.g.* Xilinx Spartan-6 FPGAs).

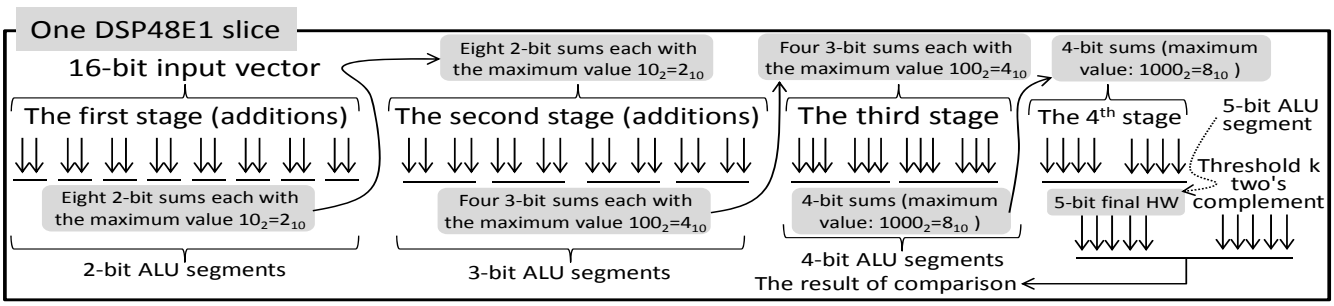

Figure 1. Computing the Hamming weight and the result of comparison with a fixed threshold for a 16-bit binary vector

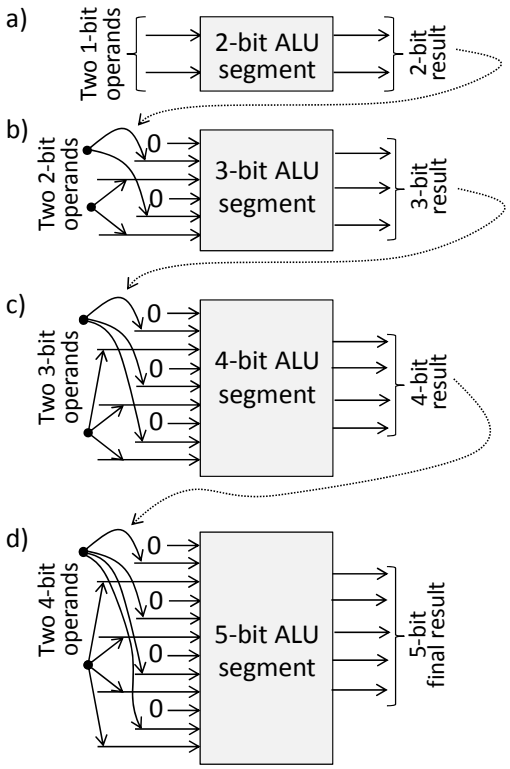

Figure 2. DSP ALU segments for implementation of tree-based additions at different stages in Fig. 1

A comparator of the HWs with a fixed threshold k can easily be built using the same DSP slice (see Fig. 1). Much like [1], the result of comparison can be obtained as HW-k, or as HW plus the 2's-complement of the threshold k on the appropriate DSP output (see Fig. 1), or if the output is occupied, in the DSP carry out line. A final comparator can also be built on an additional look-up-table (LUT) [22], enabling multiple and variable bounds to be supported.

We found that the method described permits quite complex HW counters/comparators to be implemented with moderate resources. For example, the circuit in Fig. 1 requires just one DSP48E1 slice, which can be verified using the following synthesizable VHDL code.

```
entity Test_HW16 is
  port ( Sw : in  std_logic_vector (15 downto 0);-- 16-bit vector
  -- led outputs keep the HW; led_comp gives the result of comparison with k
        led      : out  std_logic_vector (14 downto 0);
        led_comp : out std_logic);
end Test_HW16;

architecture Behavioral of Test_HW16 is
  -- A,B are DSP48E1 operands; Y keeps the result of comparison and HW
  signal A, B, Y    : std_logic_vector(47 downto 0);
  signal threshold : std_logic_vector(4 downto 0); -- fixed threshold
begin

threshold <= not "01010" + 1;  -- the threshold two's complement

process(Sw, Y, threshold)
begin
     A <= (others => '0');     -- the first 48-bit DSP operand
     B <= (others => '0');     -- the second 48-bit DSP operand

     for i in 7 downto 0 loop  -- the first stage in Fig. 1
          A(2*i) <= Sw(i);     B(2*i) <= Sw(i+8);
     end loop;

     for i in 3 downto 0 loop  -- the second stage in Fig. 1
       A(16+3*i+1 downto 16+3*i) <= Y(2*i+1 downto 2*i);
       B(16+3*i+1 downto 16+3*i) <= Y(2*i+1+8 downto 2*i+8);
     end loop;

     for i in 1 downto 0 loop  -- the third stage in Fig. 1
       A(28+4*i+2 downto 28+4*i) <= Y(16+3*i+2 downto 16+3*i);
       B(28+4*i+2 downto 28+4*i) <= Y(16+3*i+2+6 downto
                                                  16+3*i+6);
     end loop;
                                  -- the fourth stage in Fig. 1
     A(39 downto 36) <= Y(31 downto 28);
     B(39 downto 36) <= Y(35 downto 32);
     A(45 downto 41) <= Y(40 downto 36);  -- comparison
     B(45 downto 41) <= threshold;
end process;

-- the resulting Hamming weight:
led <= (14 downto 5 => '0') & Y(40 downto 36);
        -- the result of the Hamming weight comparison:
led_comp  <= Y(46);        -- see also Fig. 1

DSP: entity work.DSP48E1_HW16 -- link with the DSP slice
        port map (A, B, "0000", Y); --"0000" is the addition mode

end Behavioral;
```

A template for the DSP48E1 slice [21] that is available in the Xilinx design environment has been customized to block the multiplication operation and to assign the latency to 0. Two operands are used as ALU inputs applying concatenation operation for one operand. The mode of the ALU is set to addition [21]. Since latency is 0, the clock signal is not used.

VHDL code for N=32 is built based on the Test_HW16 entity and it has the structure shown in Fig. 3. Two Test_HW16 components compute the HW for the first and for the second 16-bit sub-vectors of the 32-bit vector. Since HW comparators are no longer needed for the sub-vectors, a part of each DSP slice becomes vacant (see the unused section in Fig. 3) for input operands $A_{41},\ldots,A_{47},B_{41},\ldots,B_{47}$. Thus, the two Test_HW16 entities resources that are released may now be used for:

1. Adding HWs of the two 16-bit sub-vectors to get the HW of the entire 32-bit vector. This is done in the first Test_HW16 component (see the upper block in Fig. 3) in which a 6-bit sum of two 5-bit HWs is produced.
2. Computing the result of comparing the 32-bit HW with a fixed threshold k (see the second Test_HW16 component at the bottom of Fig. 3).

Similarly, HW counters/comparators can be built for larger values of N. Let us discuss now how multiple and variable bounds can be supported. Since a LUT(n,1) with n inputs and 1 output can implement any Boolean function of n variables, various number of bounds for $0 \le N \le 2^n-1$ may be handled. If N is greater (or even significantly greater) than 2^n-1, the LUT(n,1) circuit can be built using the method [22] (see the circuits in Fig. 2 in [22]). Of course, embedded memory blocks may also be used. For the majority of contemporary FPGAs, such blocks with $9 \le n \le 15$ are available. Thus, the maximum value of N can be increased up to $2^{15}-1$, which enables requirements for practical cases to be satisfied. Since all memories (both distributed or LUT-based and embedded) are run-time configurable, the circuits

described above are not threshold-dependent (*i.e.* they may be dynamically customized for any value of the threshold k < 2^n-1 or for any range 0≤N≤2^n-1 for fixed or variable bounds).

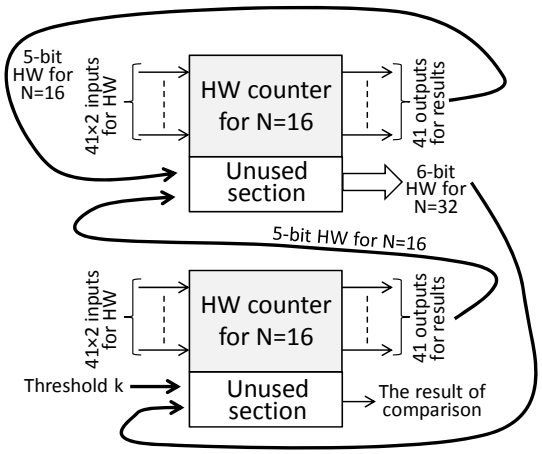

Figure 3. The structure of HW counter/comparator for N=32

IV. Implementation, Experiments, and Comparisons

This section presents a thorough evaluation and comparison of the proposed circuits that have been synthesized and implemented in the Xilinx ISE 14.7/Vivado 2014.1 from specifications in VHDL and tested in the Nexys-4 prototyping board with the recent Artix-7 FPGA xc7a100t-3csg324 [20] from the Xilinx 7 series. Some of the experiments were done in the ZedBoard [27] and ZyBo [28] with Zynq all programmable systems-on-chip (APSoCs).

The size N of input vectors was chosen from 16 to 2048 which is appropriate for the majority of practical cases. Values of N larger than 2048 are indeed exceptional, but they can also be handled because the most advanced devices contain a significantly greater number of DSP slices than the low-cost FPGAs that were used. Even in the chosen FPGAs, HW for N=16384 can be computed if we combine DSP and general-purpose logical slices. The following three methods were used to supply initial vectors: 1) from a random number generator implemented in the same FPGA; 2) from an embedded processing system (PS), such as that available in the Xilinx APSoCs; 3) from the keyboard as we will explain later in this section.

Table 1 below presents the results of synthesis, implementation and test of HW counters/comparators in the Nexys-4 board, where N is the size of vectors in bits, N_{DSP} is the number of DSP slices occupied, and D_{max} is the maximum combinational path delay in ns. N_s is the number of logical slices used (N_s=0 for all circuits in Table 1).

Table 1. The results of experiments (only DSP slices were used)

N	16	32	64	128	256	512	1024	2048
N_{DSP}	1	2	4	9	17	34	68	136
D_{max}	2.1	3.9	5.7	6.5	7.5	9.3	10.9	14.2

Table 2 presents results similar to Table 1 but DSP slices were used only up to N=64; then a set of adders built from FPGA logical slices perform the additions of HWs for all the sub-vectors with N=64. For example, to find the HW for N=512, the HWs of 512/64 = 8 sub-vectors are added. Thus, depending on the requirements and the availability of FPGA resources, different methods may be chosen.

Table 2. The results of experiments (DSP and logical slices were used)

N	128	256	512	1024	2048
N_{DSP}	8	16	32	64	128
N_s	2	6	15	33	67
D_{max}	7.0	8.6	9.8	10.9	12.6

We found that DSP-based implementations are generally faster. Only in one case for N=2048 the circuit from Table 1 is slower, which can be explained by different routing overheads.

The results of [22] demonstrate that the parallel counters, counting networks and circuits based on mapping [11] to LUTs are the fastest and the least resource consuming compared to other known methods (particularly [2,24,25]). Fig. 4 shows the maximum combinational path delays for the proposed and the best known designs. Only circuits for up to 1024-bit vectors are compared. This is because we could not find any published result for implementations of HW counters and comparators in FPGAs with N>1024.

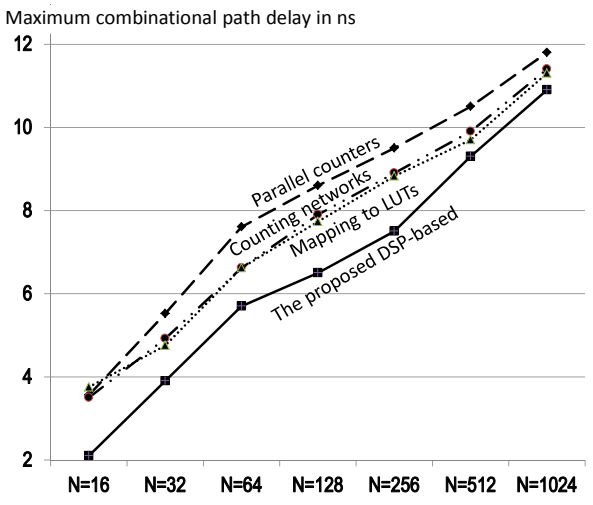

Figure 4. Comparison of the maximum combinational path delays from the results of experiments in the Nexys-4 board

Fig. 5 shows the values of N_{DSP} and N_s. Note that the numbers of DSPs and logical slices are not given for comparison with each other. The main objective is to demonstrate how effectively potential alternative FPGA resources may be used.

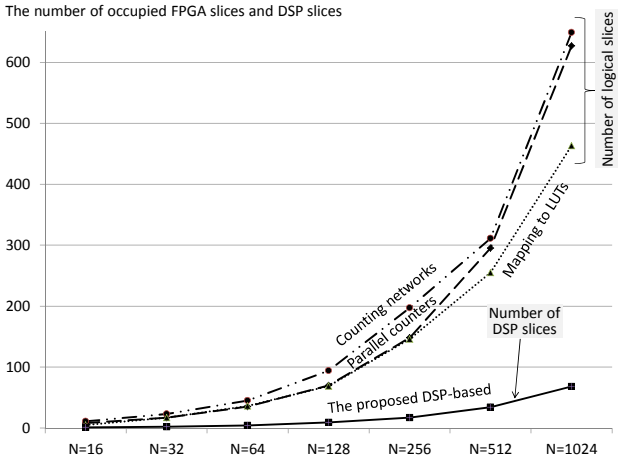

Figure 5. The occupied resources from the experiments in Nexys-4 board

It is clear from the results of experiments that the proposed solutions consume reasonable resources (see Fig.

5) and are faster (see Fig. 4) than the best known alternatives. From Tables 1 and 2 we can see that effective throughput may exceed 50 million 2048-bit vectors per second, which is significantly better that the results of benchmarks for software running in general purpose computers [16].

The HW counters and comparators described can be used as hardware accelerators in APSoCs such as the Zynq-7000. Note that the speed that is achievable in the proposed circuits is limited by the communication mechanisms available with a host computer/processor (such as the ARM Cortex-A9 in Zynq) that may use the results. A detailed research report on communication overheads in APSoCs can be found in [29].

One example of a potential application is shown in Fig. 6. The PS (*e.g.* the ARM Cortex-A9 in Zynq microchips) prepares initial data and transfers them to the external DDR memory, which is also accessible from the programmable logic (PL) located on the same microchip.

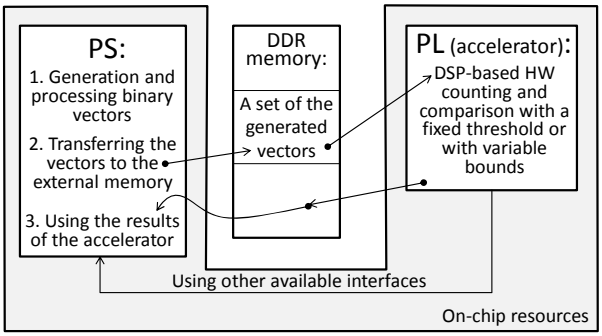

Figure 6. Potential application for an on-chip system

The PL that implements the proposed DSP-based HW counter/comparator, reads a vector (or a set of vectors) from the DDR and produces the result, which is returned to the PS. Interaction with the on-chip DDR controller is provided through Advanced eXtensible Interface (AXI), which is very fast.

Experiments just in the PL permit, in particular, hardware accelerators to be preliminary evaluated and the circuits to be tested before their integration in more complicated systems, including software and hardware components. The technique used is verified in a project that includes a VGA controller, a keypad controller and the proposed DSP-based Hamming weight counter. The Digilent keypad [30] was used to enter hexadecimal numbers that are shifted in long (2048 hexadecimal digits in the example) binary vector as follows: digits <= digits(1 **to** 2047) & Decode, where Decode is a digit from the keypad. Thus, each element of the signal digits is a hexadecimal number taken from the keypad. Even in a simple VGA monitor with resolution 800×600 pixels, more than 5000 hexadecimal numbers may be shown, allowing the functionality of Hamming weight counters for binary vectors with more than 20 thousand bits to be evaluated. The following VHDL process is used to extract individual bits from the vector that was entered:

```
process(set2048digits)
begin
  for i in 2047 downto 0 loop
    in2048bit(i) <= set2048digits(i)(0); -- bits 0 are extracted
  end loop;
end process;
```

where the in2048bit signal is declared as: **signal** in2048bit : std_logic_vector(2047 **downto** 0); and set2048digits(i)(0) is the bit with index zero (0) in the hexadecimal number with index i. Similarly, other bits (*i.e.* 1, 2, 3) can be extracted and the resulting long vectors further processed. In this case the process above is changed as follows:

```
process(set2048digits)
begin -- the vector in8192bits is std_logic_vector (8191 downto 0);
  for i in 2047 downto 0 loop
      in8192bits(i)      <= set2048digits(i)(0);
      in8192bits(i+2048) <= set2048digits(i)(1);
      in8192bits(i+4096) <= set2048digits(i)(2);
      in8192bits(i+6144) <= set2048digits(i)(3);
  end loop;
end process;
```

Thus, long binary vectors can be processed without the need for a very large number of external pins.

Fig. 7 shows how the circuit is used for visual tests. Data are typed from a mini keypad [30]. Different bits of the entered hexadecimal numbers are taken to form a binary vector (see the last process above). The resulting HW is displayed and can be evaluated preliminarily. Physical delays in the circuits may easily be found and displayed. The designed and tested circuit permits HW for N=8192 to be computed in the ZyBo (microchip Zynq xc7010-1clg400) [28] in about 1 microsecond.

Figure 8 demonstrates the utilization of resources from the post implementation report (Vivado 2014.1). As you can see, 71% of LUTs were used because at the first stages (see Fig. 1) 32-bit HW counters were constructed from LUTs using the circuits from [11], and DSPs computed the final HW from N/32 products of the LUTs.

Figure 7. Counting the HW in pure combinational DSP-based circuit implemented and tested in the PL section of Zynq 7010 microchip (ZyBo [28])

This approach permits the size of vectors to be increased further. Finally the tested projects enable HWs to be found in fully combinational circuits implemented in the PL for N=8192 in ZyBo and N=16384 in ZedBoard (Xilinx APSoC xc7z020clg484-1).

Resource	Utilization	Available	Utilization %
FF	8336.0	35200.0	24
LUT	12509.0	17600.0	71
I/O	28.0	100.0	28
BRAM	2.0	60.0	3
DSP48	68.0	80.0	85
BUFG	1.0	32.0	3
MMCM	1.0	2.0	50

Zynq xc7010-1clg400

Figure 8. Utilization of resources from the Vivado 2014.1 report (BRAM – block RAM used for VGA controller; I/O – inputs/outputs, BUFG – Xilinx buffers, MMCM – mixed-mode clock manager, FF – flip flops)

We compared the results in hardware with the results in software [31] running in the PS of the same APSoC for ZyBo. The clock frequency of the PS is 670 MHz. For N=8192, the HW is found in software in about 14 thousands processor cycles for the best program from [31]. Thus, the hardware implementation described is faster by a factor of about 20. Communication overheads have been then measured for one (of the 4 available) high-performance AXI port. We found finally that hardware is faster by a factor of about 7, including the communication overheads. The comparisons were done using timer functions from [32].

V. Conclusion

High-performance Hamming weight counters and comparators are frequently required in both software and hardware implementations. The paper suggests a novel technique that allows very fast circuits to be developed based on embedded digital signal processing slices that are widely available in contemporary FPGAs. The results of experiments clearly demonstrate that the proposed solutions require very reasonable resources and are faster than the best known alternatives.

Acknowledgment

The authors would like to thank Ivor Horton for his very useful comments and suggestions.

References

[1] 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, 2009, pp. 167-171.

[2] V. Pedroni, “Compact Hamming-comparator-based rank order filter for digital VLSI and FPGA implementations,” in Proc. IEEE Int. Symp. on Circuits and Systems, vol. 2, Canada, 2004, pp. 585-588.

[3] 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, 1989, pp. 785–794.

[4] P.D. Wendt, E.J. Coyle, and N.C. Gallagher, “Stack filters,” IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 34, no. 4, 1986, pp. 898-908.

[5] M. Storace and T. Poggi, “Digital architectures realizing piecewise-linear multivariate functions: two FPGA implementations,” Int. Journal of Circuit Theory and Applications, vol. 39, no. 1, 2011, pp. 1-15.

[6] K. Asada, S. Kumatsu, and M. Ikeda, “Associative memory with minimum Hamming distance detector and its application to bus data encoding,” in Proc. IEEE Asia-Pacific Application-Specific Integrated Circuits Conf., Korea, 1999, pp. 16-18.

[7] C. Barral, J.S. Coron, and D. Naccache, “Externalized fingerprint matching,” in Proc. Int. Conf. on Biometric Authentication, Hong Kong, 2004, pp. 309–315.

[8] A. Zakrevskij, Y. Pottosin, and L. Cheremisiniva, Combinatorial Algorithms of Discrete Mathematics, TUT Press, 2008.

[9] I. Skliarova and A.B. Ferrari, “A Software/reconfigurable hardware SAT solver,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 4, 2004, pp. 408-419.

[10] D.E. Knuth, The Art of Computer Programming, vol. 3: Sorting and Searching, Addison-Wesley, 2011.

[11] V. Sklyarov and I. Skliarova, “Digital Hamming weight and distance analyzers for binary vectors and matrices,” Int. Journal of Innovative Computing, Information and Control, vol. 9, no. 12, 2013, pp. 4825-4849.

[12] J.D. Davis, Z. Tan, F. Yu, and L. Zhang, “A practical reconfigurable hardware accelerator for Boolean satisfiability solvers,” in Proc. 45th ACM/IEEE Design Automation Conf., USA, 2008, pp. 780-785.

[13] D. Cullina, A.A. Kulkarni, and N. Kiyavash, “A coloring approach to constructing deletion correcting codes from constant weight subgraphs,” in Proc. of IEEE Int. Symp. on Information Theory, UK, 2012.

[14] Intel, Corp., Intel® SSE4 Programming Reference, 2007. [Online]. Available: http://home.ustc.edu.cn/~shengjie/REFERENCE/sse4_instruction_set.pdf

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

[16] Dalke Scientific Software, LLC, Faster population counts, 2011. [Online]. Available: http://dalkescientific.com/writings/diary/archive/2011/11/02/faster_popcount_update.html

[17] R. Ramanarayanan, S. Mathew, V. Erraguntla, R. Krishnamurthy, and S. Gueron, “A 2.1GHz 6.5mW 64-bit Unified PopCount/BitScan Datapath Unit for 65nm,” in Proc. 21st Int. Conf. on VLSI Design, India, 2008.

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

[19] 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.

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

[21] Xilinx, Inc., 7 Series DSP48E1 Slice User Guide, 2013. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[22] V. Sklyarov and I. Skliarova, “Design and implementation of counting networks,” Computing, 2013. [Online]. Available: doi: 10.1007/s00607-013-0360-y

[23] V. Sklyarov and I. Skliarova, Parallel Processing in FPGA-based Digital Circuits and Systems, TUT Press, 2013.

[24] S.J. Piestrak, “Efficient Hamming weight comparators of binary vectors,” Electronic Letters, vol. 43, no. 11, 2007, pp. 611–612.

[25] V.A. Pedroni, “Compact fixed-threshold and two-vector Hamming comparators,” Electronic Letters, vol. 39, no. 24, 2003, pp. 1705–1706.

[26] R. Mueller, J. Teubner, and G. Alonso, “Sorting Networks on FPGAs,” The Int. Journal on Very Large Data Bases, vol. 21, no. 1, 2012, pp. 1-23.

[27] Avnet, Inc., ZedBoard (Zynq™ Evaluation and Development) Hardware User’s Guide, 2013. [Online]. Available: http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v1_9.pdf

[28] Digilent, Inc., ZyBo Reference Manual, 2014. [Online]. Available: http://digilentinc.com/Data/Products/ZYBO/ZYBO_RM_B_V6.pdf

[29] M. Sadri, C. Weis, N. Wehn, and L. Benini, “Energy and Performance Exploration of Accelerator Coherency Port Using Xilinx ZYNQ,” in Proc. 10th FPGAworld Conf., Copenhagen and Stockholm, 2013.

[30] Digilent, Inc., PmodKYPD™ Reference Manual, 2011. [Online]. Available: http://digilentinc.com/Products/Detail.cfm?NavPath=2,401,940&Prod=PMODKYPD

[31] S.E. Anderson, Counting bits set, in parallel. [Online]. Available: http://graphics.stanford.edu/~seander/bithacks.html

[32] Xilinx, Inc., OS and Libraries Document Collection, 2010. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/oslib_rm.pdf