Transactions Briefs

Sorter Based Permutation Units for Media-Enhanced Microprocessors

Giorgos Dimitrakopoulos, Christos Mavrokefalidis, Kostas Galanopoulos, and Dimitris Nikolos

***Abstract*—Single or multibit subword permutations are useful in many multimedia and cryptographic applications. Several specialized instructions have been proposed to handle the required data rearrangements. In this paper, we examine the hardware implementation of the powerful permutation instruction group (GRP). The design of the proposed permutation unit is based on the functionality of sorting networks. Two variants of the sorter-based GRP unit are introduced and analyzed and their energy-delay behavior is investigated using static CMOS implementations in a 130-nm CMOS technology.**

***Index Terms*—Cryptography, data-rearrangement instructions, multimedia processors, permutation units, sorting networks.**

I. Introduction

Multimedia applications constitute a large and increasing percentage of general purpose computing workload [1], [2]. One of the main characteristics of multimedia applications is that they deal with low precision data that exhibit high levels of data parallelism. In most cases, multimedia data are packed into subwords of 1 or 2 bytes that are processed in parallel in word oriented processors according to the single instruction multiple data (SIMD) paradigm [3], [4]. New instructions have been introduced to the instruction set of modern microprocessors, in order to efficiently handle subword operations and enhance the performance of software implemented multimedia algorithms. In order to fully exploit the subword parallel operations, the subwords need to be efficiently rearranged inside the registers in order to enhance the computation. Efficient handling of permutations is also needed for the software implementation of cryptographic algorithms in order to achieve the required throughput. The selection of efficient permutation instructions and the design of fast permutation units have recently attracted a lot of interest [5]–[9].

Several instructions have been proposed for speeding up the execution of arbitrary bit permutations [5]. Among them, instruction GRP needs $\log_2 n$ instructions to generate an arbitrary permutation of n bits, assuming that at most n control bits are available for each operation. GRP has a versatile use and achieves greater speedup when used in cryptographic algorithms [5], [9]. GRP R_D, R_S, R_C takes two source operands, the data and the control bits stored in R_S and in R_C, respectively, and generates one result that is stored in the destination register R_D. The instruction divides the bits of R_S in two groups based on the control bits of R_C. If a control bit is 1, then the corresponding data bit of R_S is put in the first group. Otherwise, the bit of R_S is put in the second group. In the result, the relative position of the bits in each group remains unchanged. An example of the execution of GRP is shown in Fig. 1. Hardware implementations of GRP have already been presented in [10] and [11].

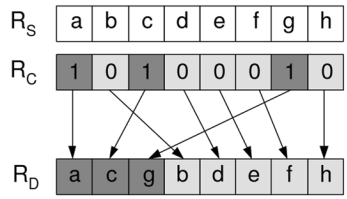

Fig. 1. Example of the execution of GRP.

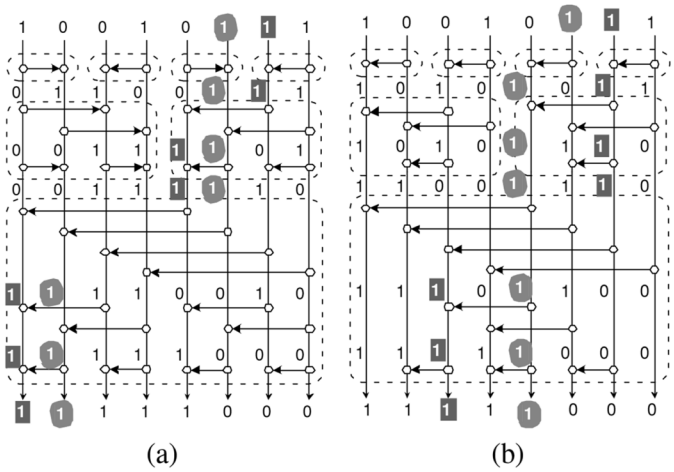

Fig. 2. 8-bit sorter designed using (a) a bitonic and (b) a merge-sorting network.

In this paper, new hardware implementations of the GRP instruction are introduced. In Section II, we present the main idea behind our work, showing that the proposed permutation units can follow a structure similar to sorting networks. In the following sections, we explore the architecture and the circuit implementation of two alternatives of the enhanced sorting networks that we propose for the design of the GRP execution unit. The efficiency of the proposed circuits has been validated using static CMOS implementations in a 130-nm CMOS technology. Experimental results are given in Section V and conclusions are drawn in Section VI.

II. Main Idea

According to GRP, the data bits that are associated with a control bit equal to one, are concentrated to the left side of the output. This action resembles a sorting operation for the control bits, where the largest bits, i.e., bits equal to one, are gathered to the left. Therefore, the problem of designing a hardware unit that executes GRP, is equivalent to the design of a sorting network that sorts the control bits and simultaneously exchanges the positions of the corresponding data bits appropriately. The main problem that arises is, that apart from sorting the control bits, we must also ensure that the relative position of the data bits remains unchanged. We consider two cases of sorting networks [12], the bitonic and the merge-sorting network, which are shown in Fig. 2 for the case of 8 bits. Each sorting network is composed of the same compare element and the direction of its arrow denotes the final position of the maximum input. In Fig. 2 both networks sort the same binary word. This binary word corresponds to the control bits of the GRP instruction. We need to examine if, at the sorted output, the relative significance of the bits inside the two groups of ones and zeros, respectively, is preserved.

Manuscript received May 9, 2006; revised November 22, 2006. This work was supported by the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the program PYTHAGORAS.

The authors are with the Technology and Computer Architecture Laboratory, Computer Engineering and Informatics Department, University of Patras, Patras 26500, Greece (e-mail: dimitrak@ceid.upatras.gr).

Digital Object Identifier 10.1109/TVLSI.2007.898750

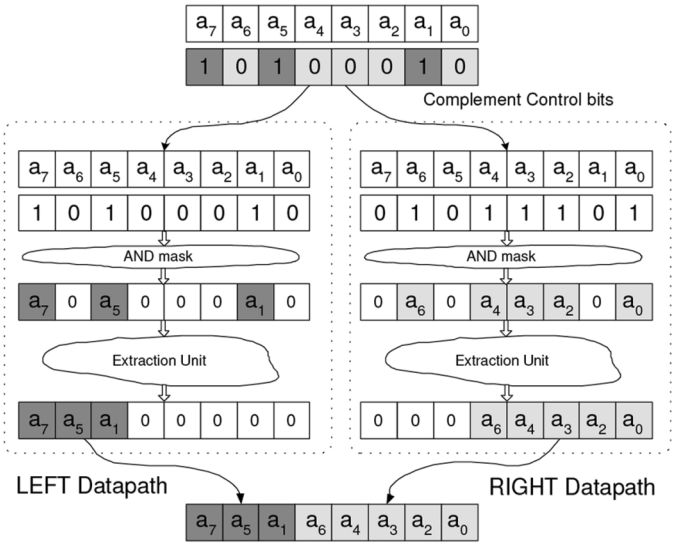

Fig. 3. Implementation of GRP using two separate extraction units.

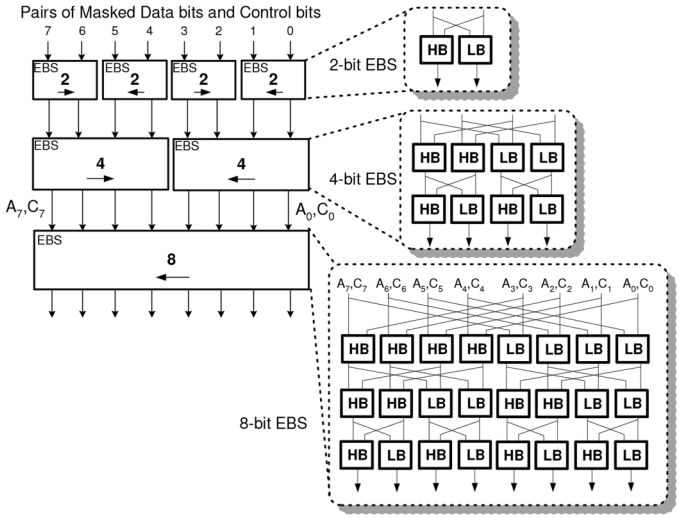

Fig. 4. Block diagram of an 8-bit enhanced bitonic-sorting network, showing also the structure of the corresponding 2-, 4-, and 8-bit EBS units.

At first, we consider the case of a bitonic-sorting network [see Fig. 2(a)]. The circuit is composed of appropriately connected subnetworks, called bitonic sorters [13]. A bitonic sorter can only sort bitonic sequences. A string of bits is a bitonic sequence, when it has the form $0^*1^*0^*$ or $1^*0^*1^*$. At each stage, two bitonic sequences are merged and a new double size bitonic sequence is produced. Bitonic sorting networks cannot preserve at the output the relative significance of equal bits. When two bitonic sequences are merged the bits are recursively divided in two independent halves. Therefore, when a bit is put in one half, it cannot regain its relative significance compared to the rest bits with the same value that were placed to the other half. Please notice the example of the two marked ones in Fig. 2(a). Although they are correctly sorted at the output (no zero bit exists in a more significant position), the routes that they followed have altered their relative significance. Therefore, if their associated data bits followed them, while they were sorted, they would be in wrong order according to the definition of GRP.

The merge sorting network, shown in Fig. 2(b), follows a different sorting principle. At each stage, two already sorted sequences, i.e., 1^*0^*, are merged to form a double size sorted sequence. The merging of the two sequences is carried out by the merge sorter. The merge sorter has the same number of compare levels as the bitonic sorter but requires less compare elements. A clear explanation for the construction of bitonic and merge sorting networks can be found in [14]. Again, as shown by the example of Fig. 2(b), the merge sorting network cannot preserve at the output the relative significance of equal bits.

Since the bitonic and merge networks cannot preserve the relative significance of equal bits, the data associated with control bits equal to one and zero, respectively, are separately extracted from the input operand and aligned at the output. The general form of the two-datapath architecture is shown in Fig. 3. The left datapath is responsible for concentrating the data bits with a control bit equal to one to the left side of the result register. In the same manner, the right datapath that assumes complemented control bits, concentrates the rest data bits to the right side of the result register. The partial results of the two extraction units are unified with a logical OR operation. In order to allow the OR unification at the output, the input data bits are first masked with the corresponding control bits. The general architecture that uses two separate extraction units was also followed in previous GRP implementations [10], [11]. However, each extraction unit follows a completely different design principle and, as we will show in Section V, it leads to the less efficient circuits compared to the proposed sorter-based GRP units.

III. Enhanced Bitonic-Sorting Network

In the following, we derive a new structure, called *enhanced bitonic-sorting network* (EBSN), that properly aligns the data bits with control bits equal to one to the left side of the output (left extraction unit). Then, following the two-datapath architecture of Fig. 3, a complete GRP unit can be easily derived. The general structure of an 8-bit EBSN is shown in Fig. 4. The EBSN takes two n-bit inputs; the data bits that need to be correctly aligned at the left side of the output, and the control bits that denote, which data bits should be selected. According to the architecture of Fig. 3, the data bits are at first masked with their associated control bits. Since we are not interested in the masked data bits that are set to zero, they are graphically represented as x values. The EBSN is composed of appropriately connected subnetworks called *enhanced bitonic sorters* (EBS). Each EBS has two sets of inputs. A set of control bits that should be in bitonic form and their associated data bits. The purpose of the EBS is to sort the control bits and appropriately move the corresponding data bits. Every two neighbor EBS units of Fig. 4 are of equal size and have opposite converging directions. This is required so as the newly derived double-size sequence of control bits, i.e., the combination of the outputs of two EBS units, to be in bitonic form.

An EBS is effectively a butterfly network. The structure of an 8-bit EBS unit that sorts the maximum elements to the left, is shown in Fig. 4. The first role of the EBS unit is to sort the bitonic sequence of control bits. When two control bits are compared, their maximum is given by their boolean OR function, which appears at the output of the HB cell. Similarly, their minimum is given by their boolean AND function and is produced at the output of the LB cell. The other purpose of the EBS unit is to move appropriately the data bits with control bits equal to one. Since the control bits are in bitonic form, we can use only a subset of them in order to align the data bits. To see this we assume for the 8-bit case that the data and control bits are equal to $xxCDExxx$ and 000111000, respectively. If we had an EBS unit that could correctly move the data bits, C, D, E, with control bits equal to one, to the left side of the output, then the result would be equal to $CDExxxxx$. We do not care about the relative significance of the x bits, since they are equal to zero. We can get the same result if we assume that all the $n/2$ less significant control bits are equal to one. Therefore, only the $n/2$ more significant of the input control bits are required to move appropriately the data bits. The rest can be safely assumed equal to one.

Following the control bits' simplification, we will describe how the data bits are moved to the output of the EBS via the 8-bit example

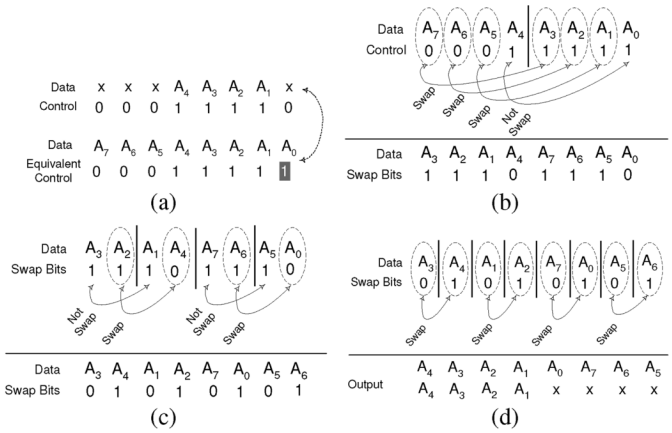

Fig. 5. Example of the functionality of the 8-bit EBS.

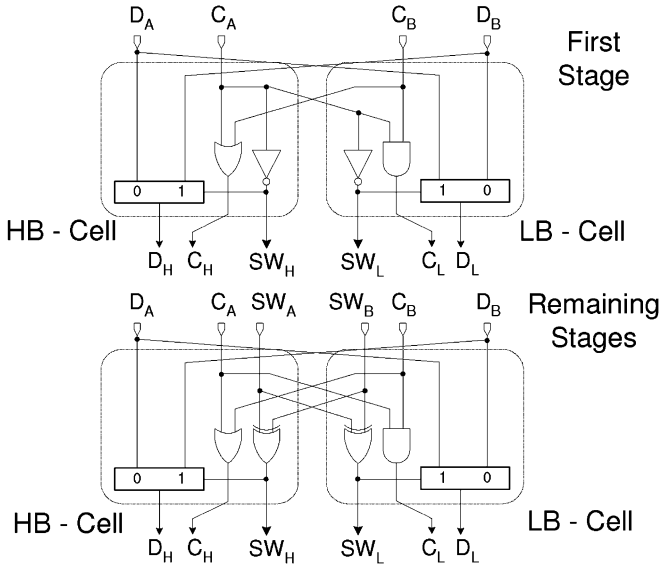

Fig. 6. Implementation of the HB and LB cells.

shown in Fig. 5. The input data and control bits are shown in Fig. 5(a). Data bits A_7, A_6, A_5, and A_0 are x values but we keep their names for clarity. Also, as explained earlier, the control bit of A_0 can be safely assumed equal to one. The connections shown in Fig. 5(b)–(d) represents the connections of the first, the second, and the third level, respectively, of the 8-bit EBS, shown in Fig. 4.

The functionality of the first stage of the 8-bit EBS is shown in Fig. 5(b). The control bits that are equal to zero at the left end of the control word, represent empty positions that must be filled by the most significant data bits of the right half. Data bits A_3, A_2, and A_1 should be exchanged with A_7, A_6, A_5, that have control bits equal to zero, in order to fill the left half of the data word. In this way, although A_3, A_2, and A_1 have moved to the correct half of the output, their relative significance with respect to A_4, that also has a control bit equal to one, has been violated. At the output of the first stage, a new control bit, called *swap* bit, is generated for each data bit. Each swap bit indicates if the corresponding data bit has changed its position at the previous stage. Hence, swap bits equal to one are assigned to data bits A_3, A_2, A_1 and A_7, A_6, A_5. The swap bits associated with A_4 and A_0 are set to zero, because they did not change their position, and they are already at the correct half of the result. In general, in the first stage, an exchange between data bits $A_{j+n/2}$ and $A_j, 0 \leq j < n/2$ takes place, only when the corresponding control bits $C_{j+n/2}$ of the upper half and C_j of the lower half are equal to 0 and 1, respectively. Since all the control bits of the lower half are assumed equal to one, the condition for an exchange reduces to $C_{j+n/2} = 0$. Hence, the newly generated swap bits are both set equal to $\overline{C}_{j+n/2}$. The implementation of the HB and LB cells used in the first level of every EBS is shown in Fig. 6. The AND/OR gates are used for sorting the initial control bits and the multiplexers that select between the corresponding data bits, are driven by the generated swap signals.

In the following, each stage uses the swap bits generated at the output of the previous stage to correct the relative position of the data bits. When the swap bits of two data bits are different, it means that their relative position is not correct, and they should be exchanged. For instance, at the second stage [see Fig. 5(c)], data bits A_2, A_4, and A_6, A_0 have different swap bits. A_2 came from the lower half of the previous stage and passed over A_4 that did not move at the first stage. Hence, an exchange should take place between A_4 and A_2 in order to correct their relative significance. At the right half of the second stage, A_6 and A_0 also have different swap bits. A_6 came from the left half of the previous stage and it was placed to a more significant position compared to A_0. Since A_6 has been swapped in the previous stage, it means that it was compared to a data bit with a larger control bit, while A_0 was compared to a data bit with an equal control bit (equal to one). Thus, A_0 should be in a more significant position than A_6 in the final result. Therefore, an exchange between A_6 and A_0 should take place in order to correct their relative position. At the output of the second stage, swap bits equal to one are assigned to the data bits that changed position at this stage. The remaining stages work in the same way due to the recursive nature of the butterfly network. Fig. 5(d) shows the last stage of the example, where data bits with different swap bits are exchanged. Since the last four data bits $A_0A_7A_6A_5$ are from the beginning equal to zero [see Fig. 5(a)], the output is correct according to the functionality of the left extraction unit.

In general, when the swap bits of two data bits are different, an exchange takes place and their new swap bits are set to one. Therefore, every new swap bit is the exclusive-OR of the swap bits of the previous stage. The HB/LB cells that implement the remaining levels of the EBS are shown in Fig. 6. The complete enhanced bitonic sorting network [see Fig. 4] requires EBS units that can sort also the data and the control bits to the right direction. In order to configure the EBS to gather the corresponding data bits to the right end of the result, just two simple changes are required. In the first stage of each EBS, the control signals of the multiplexers should be changed from $\overline{C}_{j+n/2}$ to $\overline{C}_j$. This also changes the generation of the corresponding swap bits. Finally, in all stages of the EBS, the OR gates of the HB cells should be replaced by AND gates, while the AND gates of the LB cells should be transformed to OR gates.

IV. Enhanced Merge-Sorting Network

In the case of merge sorting networks, each extraction unit is implemented using the proposed *enhanced merge-sorting network* (EMSN). The block diagram of an 8-bit EMSN-based left extraction unit is shown in Fig. 7. As in the case of EBSN, the merge-sorting network is also composed of appropriately connected subnetworks, called in this case *enhanced merge-sorters* (EMS). The purpose of the EMS unit is analogous to that of the EBS unit. However, due to the different structure of the merge-sort approach, new circuit modifications are required.

The structure of an 8-bit EMS unit is shown in Fig. 7. In the case of EMS, the corresponding cells that perform the necessary data rearrangement, are denoted as HM and LM, respectively. Also, as in the case of bitonic sorting, only the $n/2$ more significant control bits are required to guide the data bits rearrangement and the rest control bits can be assumed equal to one. The functionality of the EMS unit will be clarified using the example of Fig. 8 for the 8-bit case. The connections illustrated in Fig. 8(b)–(d) represents the connections of the 8-bit EMS unit that is shown in Fig. 7.

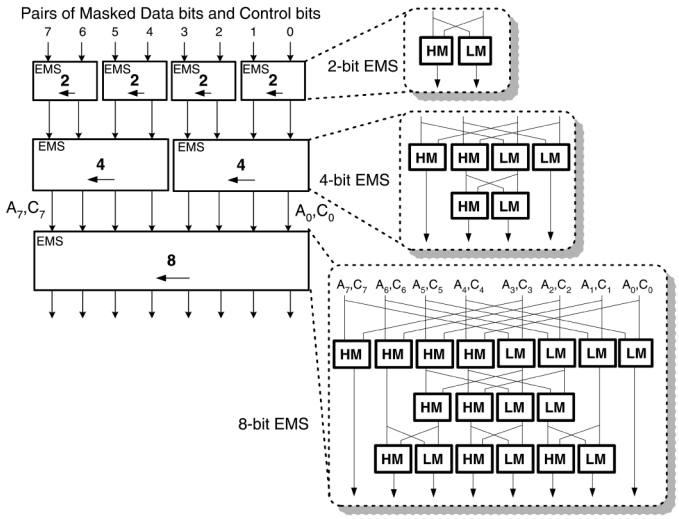

Fig. 7. 8-bit enhanced merge-sorting network and the structure of the 2-, 4-, and 8-bit EMS units.

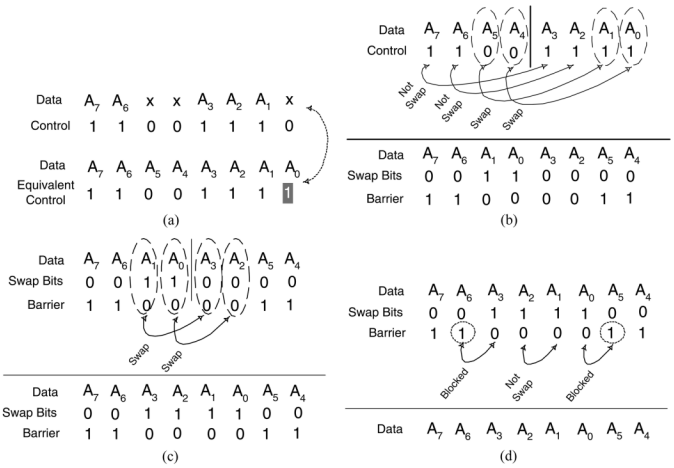

Fig. 8. Example of the functionality of the 8-bit EMS.

In Fig. 8(a), the most significant control bits of the upper part that are equal to one, suggest that no exchange should be performed at those positions. The corresponding data bits A_7 and A_6 are in the correct positions and should remain at the same positions at the output. This constraint holds for all compare levels of the EMS unit. Thus, in all levels, the HM and LM compare-and-exchange cells that receive an input from the left part of the EMS (from bit position j with $j \geq n/2$) should not perform any comparison, when the input control bit $C_j = 1$ (Barrier Constraint I). On the other hand, the zero control bits of the upper part denote empty positions that should be filled by the data bits of the lower part with control bits equal to one. Following the connections of the first level of the 8-bit EMS shown also in Fig. 8(b), the only exchanges that are allowed at the first level, are between data bits A_1, A_0 and A_5, A_4, respectively. After this operation two things occur. Data bits A_1 and A_0 have violated their relative significance with A_3 and A_2 that have control bits equal to one, while A_4 and A_5 have been correctly placed at the tail of the result. In general, an exchange between data bits $A_{j+n/2}$ and $A_j, 0 \leq j < n/2$ is performed at the first stage only when $C_{j+n/2} = 0$. Following this principle, we guarantee that the most significant data bits of the upper part will never lose their position, and the useless data bits with a zero control bit will be correctly moved to the right end of the result. In our example, all other data bits should be rearranged between A_7A_6 and A_5A_4. We should not allow any other exchange operation to be performed at bit positions 0 and 1, where A_4 and A_5 have moved.

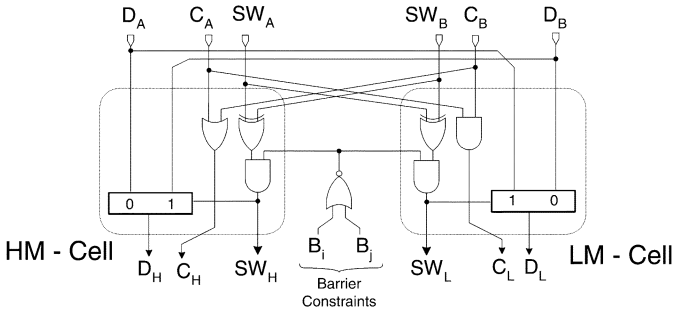

Fig. 9. Implementation of the HM/LM cell used in all stages of the EMS following the first.

In general, the number of bit positions that should be blocked in the right part of the EMS is equal to the number of zero control bits of the upper part. Therefore, in all remaining levels the HM and LM cells that receive an input from the jth bit position $j < n/2$, that belongs to the right part of the EMS, should not perform any comparison, when the input control bit $C_{j+n/2} = 0$ (Barrier Constraint II). The barrier constraints are set only by the $n/2$ more significant input control bits and block the compare-and-exchange operations of the left and the right end of the EMS, respectively.

As in the case of bitonic-sort, the data bits that have exchanged their positions at the first stage, are assigned a swap signal equal to one, while the rest, get a swap signal equal to zero [see Fig. 8(b)]. It should be noted that the swap signal generation is only required for the positions where the HM and LM cells are present in the next level. For the rest positions, it is implicitly equal to zero. For example, due to the structure of an 8-bit EMS, bit positions 7, 6, 1, and 0 directly get a zero swap signal even if an exchange has been performed at those positions.

The functionality of the remaining stages is determined by the swap signals that behave exactly the same way as in the case of the EBS. Nevertheless, for EMS, Barrier Constraints I and II should be also taken into account. An exchange is performed at the HM/LM cells, when the input swap signals are different and the barrier signals of the corresponding bit positions are both deasserted. If one of the barrier signals is asserted, no exchange is performed at the HM/LM cells. It should be noted that each swap signal is directly associated with a data bit and follows the same route in the network. On the contrary, the barrier constraints that are produced after the first level, describe a property of specific bit positions of the network and remain unchanged for all compare levels of the EMS. The barrier B_j associated with the jth bit position is equal to C_j for $j > n/2$ (left part), while in case that $j < n/2$ (right part) $B_j = \overline{C}_{j+n/2}$. Following this rule, and returning back to the example of Fig. 8(c), data bits A_3 and A_2 are exchanged with A_1 and A_0, respectively, since they have different swap signals and the operation is not blocked by any barrier constraint. The last level of the EMS network is shown in Fig. 8(d). Data bits A_2 and A_1 are correctly placed and do not move since they have equal swap signals. Also, although A_6 and A_3 have different swap signals, they are not exchanged since the barrier associated with the 6th bit position is equal to one (Barrier Constraint I, $B_6 = C_6 = 1$). The same holds for A_0 and A_5. In the latter case, Barrier Constraint II is satisfied since $B_1 = \overline{C}_5 = 1$.

The connections and the functionality of the first level of the EMS is the same with the functionality of the first level of the EBS. Therefore, the HM/LM cells of the first level are implemented in the same way as in the case of EBS and their circuit implementation is shown in Fig. 6. For the remaining levels of the EMS, the implementation of the HM/LM cells is shown in Fig. 9. The AND/OR gates that are used to sort the input control bits, are not blocked by any constraint and do not interfere with the data-exchange operation.

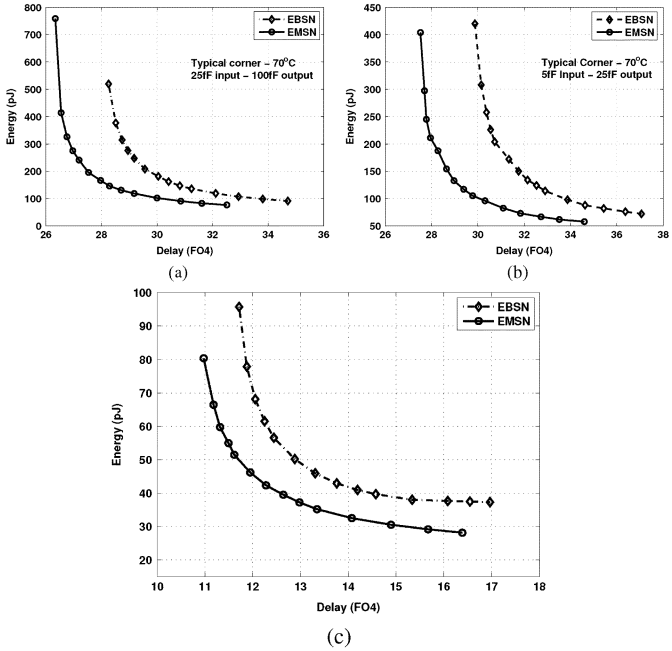

Fig. 10. Energy-delay curves for the 64-bit GRP units implemented with the EBSN and EMSN, respectively, (a) $C_{\rm in}/C_{\rm out}$ = 25/100 fF and (b) $C_{\rm in}/C_{\rm out}$ = 5/25 fF. (c) The energy-delay curve for a subword oriented GRP unit that supports a minumum subword size of 1 byte.

V. Experimental Results

The proposed circuits have been evaluated using static CMOS implementations in a 130-nm CMOS technology. The delay measurements for all examined designs are reported in fanout-of-4 inverter delays (FO4). The FO4 delay metric equals to the delay of an inverter that drives four equally-sized inverters, and it is used since it provides in some sense a technology independent way to express the delay of a circuit. In order to explore the energy-delay space for each design, we performed gate sizing for several delay targets, beginning from the circuit's minimum achievable delay. Circuit sizing is performed using geometric-programming-based optimization [15]. For the derived gate sizes, the energy and the delay of each circuit have been measured in HSpice. For the energy measurements, we assumed random inputs that caused on average 30% switching activity. Interstage wiring loads have also been taken into account. The *RC* contribution of each wire has been estimated according to its length, assuming a bit pitch of 16 metal-1 tracks.

At first, 64-bit GRP units have been evaluated. The energy-delay curves of the EBSN and the EMSN-based GRP units, under two different loading conditions and for the typical process corner, are shown in Fig. 10(a) and (b). For the data of Fig. 10(a), we assumed during optimization and measurements that the outputs of the circuit are loaded with a capacitance of 100 fF and that the maximum allowable input capacitance of each circuit is less than 25 fF, which corresponds to a ratio of 4 for the circuit's output to input capacitance. In Fig. 10(b), both circuits are optimized for lighter load, since the assumed input and output capacitance is equal to 5 and 25 fF, respectively. In both cases, the GRP unit designed using the EMSN is faster and requires less energy per operation for the same delay compared to EBSN. This result is justified from the reduced number of compare-and-exchange cells required by the EMSN, which leads to less fanout on the cells' outputs and simplifies wiring. From Fig. 10(a) and (b), it is derived that the energy required by the EMSN is less by 10%–75% compared to that of the EBSN for equal delay.

The delay of the EMSN-based GRP unit reported in Fig. 10(a) and (b) is roughly between 27 and 35 FO4. Based on the analysis presented in [11], the most efficient previous GRP implementation achieves a minimum delay of around 38 FO4 assuming equal input and output capacitances, which, under heavy output loading conditions, is rather unrealistic. Thus, the best of the proposed 64-bit GRP units achieves significant delay reductions compared to previous solutions that range between 8% and 28%.

The energy-delay behavior of the most practical case of a GRP unit that supports a minimum subword of 1 byte on a 64-bit word using the two variants of the enhanced sorting network, is shown in Fig. 10(c). The circuits are sized for the 25/100 fF input/output capacitance case. The EMSN-based GRP unit is again the fastest solution achieving a minimum delay of 11FO4. This delay is roughly equal to a 64-bit static CMOS Radix-2 adder implemented in the same technology. However, the energy spent per addition is significantly smaller. Again, the EBSN approach is the worst solution, since, for every delay target, requires more energy than the EMSN.

VI. Conclusion

A novel framework for the design of GRP permutation units has been presented in this paper. The functionality of GRP has been transformed to a sorting problem, and two enhanced sorting networks have been derived. Each one of the proposed circuits is designed using a single processing cell, while the connections between the cells are regular and are well suited for a dense datapath-style layout. Also, the speed/energy savings of the proposed solutions does not come from any special or tricky circuit technique that will not be viable in future technologies, but is a result of a new algorithmic and logic-level approach. Therefore, we believe that the proposed designs are scalable to any future technology.

References

[1] K. Diefendorff and P. K. Dubey, "How mutimedia will change processor design," *IEEE Computer*, vol. 30, no. 9, pp. 43–45, Sep. 1997.
[2] N. T. Slingerland and A. J. Smith, "Multimedia extensions for general purpose microprocessors: A survey," *Microprocess. Microsyst.*, vol. 29, no. 5, pp. 225–246, 2005.
[3] T. Conte *et al.*, "Challenges to combining general-purpose and multimedia processors," *IEEE Computer*, vol. 30, no. 12, pp. 33–37, Dec. 1997.
[4] I. Kuroda and T. Nishitani, "Multimedia processors," *Proc. IEEE*, vol. 86, no. 6, pp. 1203–1221, Jun. 1998.
[5] R. B. Lee, Z. Shi, and X. Yang, "Efficient permutation instructions for fast software cryptography," *IEEE Micro*, vol. 21, no. 6, pp. 56–69, Nov./Dec. 2001.
[6] X. Yang and R. B. Lee, "Fast subword permutation instructions using omega and flip network stages," in *Proc. IEEE Int. Conf. Comput. Design*, 2000, pp. 15–22.
[7] J. P. McGregor and R. B. Lee, "Architectural enhancements for fast subword permutations with repetitions in cryptographic applications," in *Proc. IEEE Int. Conf. Comput. Design*, 2001, pp. 453–461.
[8] Z. Shi and R. B. Lee, "Bit permutation instructions for accelerating software cryptography," in *Proc. IEEE Conf. Application-Specific Syst., Arch. Process.*, 2000, pp. 138–148.
[9] Z. J. Shi, "Bit permutation instructions: Architecture, implementation and cryptographic properties," Ph.D. dissertation, Electr. Eng. Dept., Princeton Univ., Princeton, NJ, 2004.
[10] Z. J. Shi and R. B. Lee, "Implementation complexity of bit permutation instructions," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, 2003, pp. 879–886.
[11] Y. Hilewitz, Z. J. Shi, and R. B. Lee, "Comparing fast implementations of bit permutation instructions," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, 2004, pp. 1856–1863.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 1990.
[13] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Joint Comput. Conf.*, 1968, pp. 307–314.
[14] Z. Hong and R. Sedgewick, "Notes on merging networks," in *Proc. ACM Symp. Theory Comput.*, 1982, pp. 296–302.
[15] S. P. Boyd, S. J. Kim, D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Oper. Res.*, vol. 53, no. 6, pp. 899–932, Nov./Dec. 2005.