Fast Bit Permutation Unit for Media Enhanced Microprocessors

Giorgos Dimitrakopoulos, Christos Mavrokefalidis, Kostas Galanopoulos and Dimitris Nikolos
Technology and Computer Architecture Lab
Computer Engineering and Informatics Dept.
University of Patras, 26500 Patras, Greece
E-mail: {dimitrak, maurokef, galanopu, nikolosd}@ceid.upatras.gr

***Abstract*— Bit and subword permutations are useful in many multimedia and cryptographic applications. New shift and permute instructions have been added to the instruction set of general-purpose microprocessors to efficiently implement the required data permutations. In this paper, the design of a high speed bit permutation unit is examined. The proposed architecture has been derived by mapping the functionality of one of the most powerful bit permutation instructions (GRP) to a new enhanced bitonic sorting network. The proposed design achieves delay reductions more than 20% when compared with previously presented solutions, while its regularity enables efficient VLSI implementations.**

I. Introduction

Multimedia processing deals with low precision data that exhibit high levels of data parallelism. Multimedia data are packed into subwords that are processed in parallel, according to the SIMD paradigm [1]. Several new instructions have been introduced in order to efficiently handle subword parallel operations. Besides parallel arithmetic operations, in many cases, the subwords need to be rearranged inside the registers to enhance the required computation. Therefore new shift and permute instructions have been proposed that handle all required data permutations in the subword level [2].

Besides multimedia processing, efficient bit permutation instructions are needed for the software implementation of block ciphers in order to achieve the required throughput [3]. Since cryptographic algorithms consume an increasing percentage of processor's workload, the selection of efficient bit permutation instructions, and the design of fast bit permutation units has recently attracted a lot of interest [4]–[7]. Bit permutations is the most difficult form of subword permutations. The difficulty lies in the large number of distinct possible results ($n!$ permutations of a n-bit value are possible). Several instructions and their hardware implementations have been proposed to efficiently perform arbitrary bit permutations [3]. Among them CROSS, OMFLIP, IBFLY, and GRP need the smallest number of instructions, i.e., $\log_2 n$, to generate an arbitrary permutation of n bits. Besides GRP, the functionality of the rest instructions is a direct mapping to a known interconnection network. Although this approach leads to efficient hardware implementations it has one major drawback. These instructions do not have any easily described functionality. In order to determine the outcome of the execution of the instruction, the behavior of the corresponding interconnection network has to be simulated. Therefore their adoption by a general purpose instruction set is limited.

The GRP instruction does not have the above mentioned limitations and achieves greater speedup when used in cryptographic applications [7]. GRP has a more general functionality and its use is versatile. The benefits of using GRP in other applications has been examined in [8]. The GRP R_d, R_s, R_c instruction takes two n-bit source operands, the data and the control bits stored in R_s and R_c, respectively, and generates one result that is put in the destination register R_d. The instruction divides the bits of the source register in two groups according to the values of the control bits. If a control bit is 1, then the corresponding data bit of R_s is put into the first group. Otherwise, the bit of R_s is put into the second group. The final relative position of the bits within each group remains unchanged. An example of the function of the GRP instruction is shown in Fig. 1.

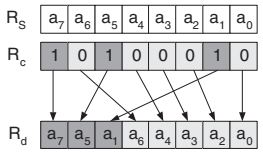

Fig. 1. Example of the GRP functionality.

The above discussion shows that GRP is the most interesting candidate instruction to be incorporated in the instruction set of a general-purpose microprocessor. In this paper a new hardware implementation for the GRP instruction is proposed. The proposed permutation unit is fast and scalable and it's design resembles the functionality of bitonic sorting networks. Also, due to its regular structure allows efficient VLSI implementation. The proposed bit permutation unit is compared to previously published designs using static CMOS implementations. The proposed design is faster achieving delay reductions more than 20%.

The rest of the paper is organized as follows. Section II gives a brief review of the previous GRP hardware implementations. Section III describes in detail the proposed architecture, while its execution latency is analyzed in Section IV. Finally conclusions are drawn in Section V.

II. Previous Work

Two circuits have been proposed for the implementation of the GRP instruction [9], [10]. In both cases, the two datapath architecture, shown in Fig. 2, is followed. The left datapath is responsible for concentrating the data bits with control bits

We thank the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program PYTHAGORAS, for funding the above work.

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 two extraction units have almost identical structure, since the direction of the concentration of the data bits slightly alters their design.

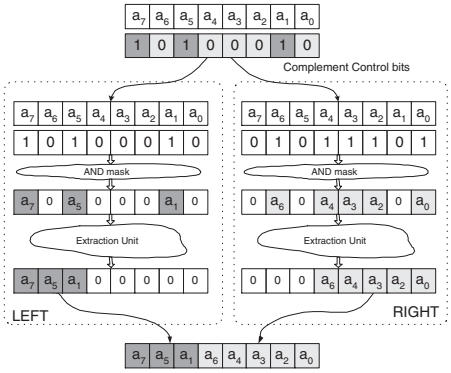

Fig. 2. The general architecture of the bit permutation unit.

The first implementation of the extraction unit [9] tries to combine consecutive groups of bits into larger groups recursively. The resulting groups contain data bits on one end and zeros on the other. At each stage the merging of two groups is performed with shift operations, and the shifting amount is determined by accumulating the number of shifting positions used in previous stages. The required shifts are encoded in one-hot code and a circuit for performing one-hot additions has been proposed. The resulting shifting amount configures a set of shifters so as to produce a larger group of properly aligned bits.

The second implementation, instead of using cascaded shifters, performs the same operation using an inverse butterfly network and a special decoder [10]. The decoder takes as input the control bits of the `GRP` and produces the bits that configure the inverse butterfly network. In the first decoding stage the number of ones in specific parts of the control bits are counted, using a tree of carry-save one's counters. The derived values are given to a set of special rotation circuits that produce the appropriate select signals for the multiplexers of the network.

The general architecture, shown in Fig. 2 is also followed by the proposed approach. However the implementation of the extraction unit is more efficient and relies on the introduced in this paper enhanced bitonic sorting networks.

III. High Speed Extraction Unit

According to the definition of the `GRP` instruction, the data bits, that have a corresponding 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.

A general sorting network is composed of appropriately connected subnetworks, called bitonic sorters, that sort a special form of bit sequences, called bitonic sequences [11]. A string of bits is bitonic when it is of the form $11\ldots100\ldots011\ldots1$ or $00\ldots011\ldots100\ldots0$. A bitonic sorter is effectively a butterfly network. Each cell of the butterfly selects either the maximum or the minimum of the two input bits. When two bits are compared, their maximum is given by the boolean OR function, while the corresponding minimum is given by the boolean AND function. Fig. 3 shows a bitonic sorter of 8 bits. In order to construct a general sorting network $\log_2 n$ stages of bitonic sorters are used. At each stage, two bitonic sequences are merged and a new double size bitonic sequence is produced [11].

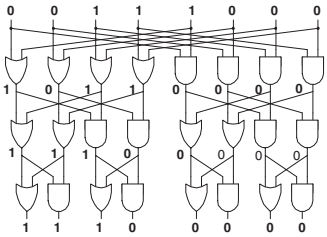

Fig. 3. An 8-bit bitonic sequence sorting network.

The basic building block of the proposed extraction unit is the *Enhanced Bitonic Sorter* (EBS). This circuit can sort any data bits preserving also their relative significance assuming that the corresponding control bits are in bitonic form.

A. Enhanced Bitonic Sorter

The enhanced bitonic sorter has two n-bit inputs. The data bits that need to be sorted to the left end of the result, and the corresponding control bits that denote which data bits should be selected. According to the construction of the extraction unit, the control bits of a n-bit EBS are bitonic sequences of the form $0\ldots01\ldots1 : 1\ldots10\ldots0$, where : denotes the middle of the two halves. For example, for a 4-bit EBS, if the data and the control bits are `[a b : c d]`, and `[0 1 : 1 0]`, respectively, the result will be `[b c : x x]` and `[1 1 : 0 0]`. The `x` values point out the fact that the EBS unit does not necessarily preserve the relative positions of the data bits with zero control bits.

The EBS is composed of two datapaths. One is responsible for sorting the control bits using a bitonic sorter as the one in Fig. 3. The other is responsible for sorting the data bits

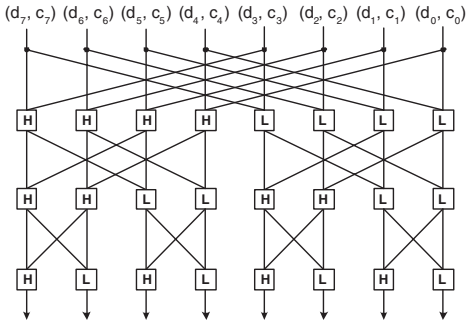

Fig. 4. The architecture of an EBS unit.

having control bits equal to one without changing their relative position. Fig. 4 shows the butterfly structure of the EBS unit that is used to sort the desired data bits. In the following we will describe the proposed algorithm and the circuit level implementation of the H and L blocks.

The butterfly network works recursively. The first stage divides the data bits into two independent halves. The interconnections guarantee that the data bits will be in the correct half at the output of the first stage (they should reside in this half at the final output). However, their relative positions might be wrong. At the second stage, data are again divided to the correct half and their relative position is partially corrected. This procedure continues, recursively, until the last stage where the final correction of the relative positions is made.

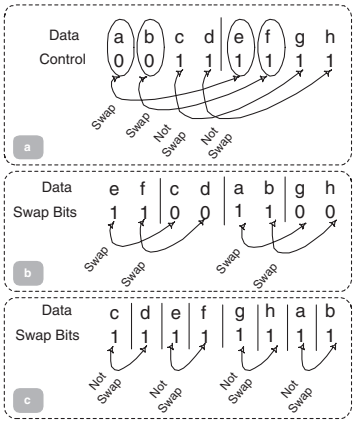

Fig. 6. Example of the proposed sorting algorithm.

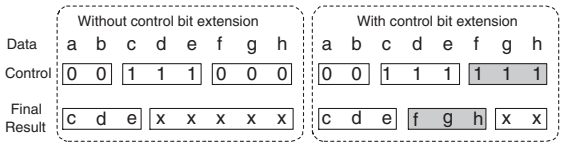

Fig. 5. Control bits extension.

In order to simplify the hardware complexity of the proposed EBS we assume that the $n/2$ least significant control bits (right half) are always equal to 1. This consideration does not alter the functionality of the EBS unit. As shown in Fig. 5, since we are only interested in sorting correctly the bits c, d, and e, even if we change the control bits of the last three data to 1 the final result remains in both cases the same. Recall that the bits in positions f, g, and h have already been nullified because of the AND masking operation in the first stage of the bit permutation unit (see Fig. 2).

Figure 6(a) gives an example of the functionality of the first stage. The bits that are equal to zero at the left-end of the control word, represent empty positions that can be filled by the most significant data bits of the right half. The data bits e and f should be exchanged with the bits a and b that have control bits equal to zero, in order to fill up the left half of the data word. At the output of the 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 the data bits e, f, a and d. The swap bits of c, d, g and h are set to zero, since they are already in the correct half of the result and hence they should not move. In general, at this stage a swap between data bits d_i and $d_{i-n/2}$ takes place only when the corresponding control bits c_i and $c_{i-n/2}$ are equal to $(0, 1)$. Therefore the newly generated swap bit s_i equals to the complement of bit c_i, i.e., $\overline{c}_i$, since each $c_{i-n/2}$ is assumed to be equal to one.

Each stage k use the swap bits generated at stage $k-1$ to correct the relative position of the data bits at each half. When the swap bits of two data bits are different it means that their relative position is not correct and they should be swapped. For instance, at the second stage (Fig. 6(b)) the data bits e-c and f-d have different swap bits. Data bits e and f came from the right half of the previous stage and passed over c and d that are more significant, i.e they should be on the left hand of data bits e and f at the final result. However they did not changed their position at the previous stage. Hence, an exchange should take place in order to correct their relative position. At the right half of the second stage, data bits a-g and b-h also have different swap bits. The data bits a and b came from the left half of the previous stage and they were placed to more significant positions compared to the data bits g and h. Because data bits g and h originally had control bits equal to one an exchange with a and b, respectively, 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 because of the recursive nature of the butterfly network. Fig. 6(c) shows the last stage of the example. Hence, when the swap bits of two data bits are different, a swap between them takes place and their new swap bits are equal to one. The datapaths for sorting the data and the control bits both rely on the same butterfly network. This means that the two butterflies can be merged, as shown in Fig. 7.

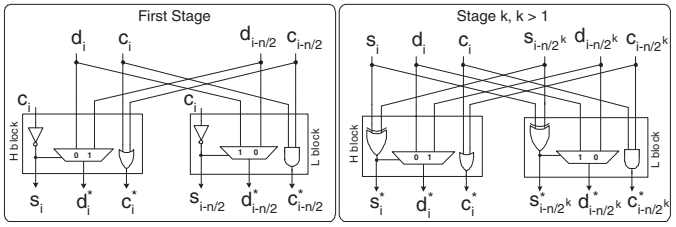

Fig. 7. The logic-level implementation of the H and L cells.

B. Sorting in the opposite direction

In order to configure the proposed EBS to gather the corresponding data bits in the opposite direction, i.e., right end of the result, just two simple changes are only required. In the first stage of the EBS unit the control signals of the multiplexers should be changed from $\overline{c}_i$ to $\overline{c}_{i-n/2}$. This also changes the generation of the corresponding swap bits. Also, the OR gates of the H blocks should be replaced by AND gates, while the AND gates of the L blocks should be transformed to OR gates.

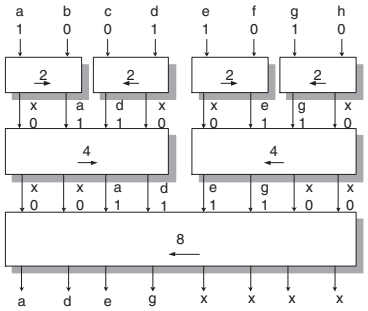

Fig. 8. The structure of a complete 8-bit extraction unit.

C. Complete Extraction Unit

A general extraction unit can sort data bits with arbitrary control bits and is implemented in $\log_2 n$ stages. At each stage i there are $n/2^i$ EBS units of size 2^i. In every stage, two neighbor EBS sort their data and control bits in opposite directions in order the resulting control bits to form a bitonic sequence. Fig. 8 illustrates an 8-bit extraction module. The number in each EBS determines its size and the arrows denote the direction of the sorting procedure. For the extraction unit of the right datapath of Fig. 2 the sorting direction of the last EBS unit should change from left to right. Also the H and L cells of the last 8-bit EBS do not require the OR/AND gates that sort the control bits since no new control bits are generated by the extraction unit. Fig. 8 also shows an example of the functionality of the circuit.

IV. Results

The proposed circuit has been evaluated using the method of Logical Effort [12]. The delay results have been obtained in a standard performance 130nm technology [13]. Characterization of the technology was performed for the typical process corner at a temperature of 70^{o}C, assuming a nominal supply voltage of 1.2V. In order to get accurate estimates of the delay of the circuit, each gate's parameters, i.e. logical effort and parasitic delay, have been derived using HSpice (see Table I).

Fig. 9 shows the delay of a 64-bit `GRP` unit for various input capacitances assuming that the outputs of the circuit are loaded with a capacitance of 300fF that roughly corresponds to a 1mm metal-2 wire. Interstage lateral wiring loads have also been taken into account, assuming a 7μm bit slice, i.e., 18 metal-1 tracks. Minimum delay gate sizing has been performed according to the method of Logical Effort. The worst input of each gate has been considered. The control bits arrive earlier than the data inputs of the multiplexers. Therefore, the critical path of the circuit goes through the multiplexers network that rearranges the data bits. Since fixed wiring loads are taken into account several iterations are required in order to perform minimum delay gate sizing.

For the cases of fast designs ($C_{out}/C_{in} < 10$) the delay of the proposed design is below 30.5 FO4 (1FO4 $\sim$ 62ps, for the technology used). According to the logical effort analysis presented in [10] the previous implementations of `GRP` had a minimum delay greater than 38.1 FO4 assuming only the case of path electrical effort of 1, which is not so realistic. Therefore it is derived that the proposed bit permutation unit is expected to be faster by at least 20%. In real case the delay savings are expected to be greater. Finally we can say that the proposed bit permutation unit can fit in two pipeline stages of a high-speed microprocessor.

TABLE I

Delay Characterization of various logic gates. Mux and Xor gates have been implemented using transmission gates.

130nm Static CMOS	INV	NAND		NOR		MUX2:1		XOR	
		A	B	A	B	In	S	A	B
Logical Effort	1	1.21	1.22	1.60	1.53	1.73	1.74	1.73	1.36
Parasitic	1.02	1.92	1.48	2.48	1.79	2.76	3.80	3.82	5.13

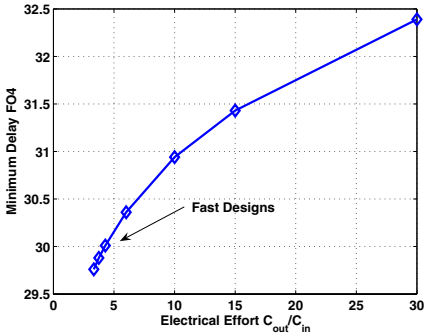

Fig. 9. The delay of the proposed 64-bit GRP unit for various ratios of input to output loading capacitance.

V. Conclusions

A new fast bit permutation unit has been proposed in this paper. New enhanced bitonic sorting networks have been introduced that allow the efficient realization of the `GRP` instruction. Taking into account the versatility and scalability of `GRP`, we conclude that media enhanced microprocessors can trully benefit by the proposed bit permutation unit.

References

[1] T. Conte et al., "Challenges to Combining General-Purpose and Multimedia Processors," *Computer*, vol. 12, no. 30, pp. 33–37, Dec. 1997.
[2] Intel, *Intel Itanium Architecture Software Developers Manual*, 2002.
[3] R. B. Lee, Z. Shi, and X. Yang, "Efficient permutation instructions for fast software cryptography," *IEEE Micro*, no. 6, pp. 56–69, Dec 2001.
[4] X. Yang and R. B. Lee, "Fast subword permutation instructions using omega and flip network stages," in *Proc. of IEEE International Conference on Computer Design*, Sept 2000, pp. 15–22.
[5] J. P. McGregor and R. B. Lee, "Architectural enhancements for fast subword permutations with repetitions in cryptographic applications," in *Proc. of IEEE ICCD*, Sept 2001, pp. 453–461.
[6] Z. Shi and R. B. Lee, "Bit permutation instructions for accelerating software cryptography," in *IEEE ASAP*, July 2000, pp. 138–148.
[7] Z. J. Shi, *Bit Permutation Instructions: Architecture, Implementation and Cryptographic Properties*. PhD Thesis, Princeton University, 2004.
[8] Z. Shi and R. B. Lee, "Subword sorting with versatile permutation instructions," in *IEEE ICCD*, Sept 2002, pp. 234–241.
[9] Z. J. Shi and R. B. Lee, "Implementation complexity of bit permutation instructions," in *Proc. of IEEE Asilomar Conference on Signals, Systems and Computers*, Nov 2003, pp. 879–886.
[10] Y. Hilewitz, Z. J. Shi, and R. B. Lee, "Comparing fast implementations of bit permutation instructions," in *Proc. of IEEE Asilomar Conference on Signals, Systems and Computers*, Nov 2004, pp. 1856–1863.
[11] K. E. Batcher, "Sorting networks and their applications," in *AFIPS proc. Spring Joint Computer Conference*, 1968, pp. 307–314.
[12] D. H. I. Sunderland, B. Sproul, *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, 1999.
[13] *UMC 130nm CMOS FSG process*. Europractice IC Service.