Parallel Multipliers Based on Horizontal Compressors

Luigi Ciminiera

Dipartimento di Automatica e Informatica
Politecnico di Torino
corso Duca degli Abruzzi, 24
10129 Torino, Italy

ABSTRACT

Two new implementations of parallel multipliers, based on iterative arrays of logic cells are presented in this paper. Both are able to compute the product of two n bit numbers with a delay of n cells, rather 2n-1 as in classical structures. The high speed operation is obtained by using pure horizontal compressors, to accelerate the horizontal signal propagation, and by adopting a suitable array structure, to shorten the vertical signal propagation. The cost and performance advantages over similar structures based on vertical compressors are discussed.

1 Introduction

The algorithms and the implementations for digital multipliers have been one of the most important research topics in the field of computer arithmetic in the last thirty years; the reason for the special attention paid for multiplication is that this operation plays an important role in almost all the arithmetic algorithms in different scientific and engineering fields. In particular, signal and image processing and algorithms of linear algebra all require the computation of a lot of multiplications.

Different types of multipliers have been presented in the literature; a possible classification may be based on whether the operands are input into the arithmetic unit in parallel or in bit-serial form. Three types of multipliers are obtained with this classification: fully serial multipliers, receiving both operands in serial form [1]-[4], serial-parallel multipliers, where one operand is input in serial form and the other in parallel [5]-[6], and parallel multipliers, where both operands are input in parallel form.

The latter class provides the fastest and most expensive solutions, and it is the class of interest in this paper. Parallel multipliers are composed by two distinct parts: an array of n^2 AND gates generating the elementary products and a circuit to add all these bits. The differences between the solutions presented in the literature are in the latter part of the multiplier.

Multipliers with irregular structures have been presented in [7]-[9]; they are aimed at minimizing the number of active circuits required for the implementation. Multipliers based on a regular array of identical cells have been introduced in [10]-[12]; full-adders are used as cells and the whole multiplier requires $O\left(n^2\right)$ cells, rather than $O(n \log n)$ cells used by irregular structures, however the iterative interconnection pattern makes them better suited for VLSI implementation.

The delay of a multiplier is due to the time required to propagate partial results both through the different weigths (carry propagation) and through the different circuits processing bits of the same weight. Owing to the usual representation adopted for the multipliers, the first type of propagation will be referred to as horizontal, and the second as vertical.

Attempts to reduce the overall multiplier delay have often been based on the use of basic arithmetic circuits able to add at once several bits arranged on different weights, and with more than one bit per weight; if such cells can be implemented with a delay close to the that of a full-adder, then the horizontal propagation delay is divided by the number of weights processed and the vertical propagation delay is divided by the number of bits per weight (approximately). These cells, called parallel counters have been proposed to be used in both iterative and non-iterative multipliers [14]-[19]. Though several parallel counters, with different distributions of the input bits, have been proposed, in general, the vertical compression has always been privileged over the horizontal one; in a recent paper [14], an iterative multiplier based on (5,3) counter, performing only vertical compression, has been presented.

This paper argues that parallel counters performing horizontal compression (with minimal vertical compression) can be used to build iterative arrays, with cost and performance characteristics better than their counterparts based on counters performing vertical compression (with minimal horizontal compression). In order to support this thesis, two arrays achieving a delay of n cells for a nxn bit multiplier are shown; both use horizontal compressors in order to shorten the horizontal delay, while only a minimal vertical compression is performed within each cell (the vertical propagation delay is decreased by using a suitable interconnection structure of the array). The construction rule of the array, leading to a short vertical delay, can be applied recursively several times, so that other multiplying structures may be obtained, requiring even bigger horizontal compressors in order to maintain the speed advantages achieved by the array structure for the vertical delay.

The paper is organized as follows. In section 2, the basic cells considered are introduced, the construction rule allowing to shorten the vertical propagation delay is derived and the first type of multiplier, based on a single type of cell is presented. In section 3, a second array based on classical full-adders and an addition circuit is presented. In section 4, the cost and performance characteristics of the proposed iterative multipliers are compared with those of classical and recent similar multiplying arrays; furthermore, the effect of the introduction of larger horizontal compressors is discussed.

2 Horizontal compressor array

The term horizontal compressor is used in this paper to indicate an

CH2419-0/87/0000/0063$01.00

arithmetic function able to add up several bits of different weight, with only few bits for each weight. If this function is implemented using a fast circuit, then it is possible to use them in order to shorten the horizontal carry propagation in arithmetic arrays.

A pure horizontal compressor is represented by an m-bit full-adder (mFA), that is an arithmetic function able to add two m-bit numbers and a carry-in bit, producing the result on m bits and a carry-out. In formulas, the mFA can be expressed as follows, where the inputs are shown on the right hand and the outputs on the left:

$$c_{out}2^m + \sum_{i=0}^{m-1} s_i 2^i = \sum_{i=0}^{m-1} (a_i + b_i)\, 2^i + c_{in} \qquad (1)$$

It is possible to see that the mFA is the horizontal counterpart of the p inputs- q outputs parallel counter, since the inputs of the latter are arranged to add the maximum number of bits per weight, with the minimal number of weights (only 1); viceversa, the inputs of an mFA are arranged to cover the maximum number of weights, with the minimum number of bits per weight to be added.

In the following, the attention will be focused on 2FA units. The rationale behind this choice is that VLSI technology does not allow the implementation of complex functions with the same speed as simple ones, even if an expensive implementation is used. This characteristic seems to be even more valid for GaAs technology, as noted in [20]. However, the use of horizontal compressor larger than 2FA will be also briefly discussed in this paper.

Before illustrating a quasi iterative multiplying array based on 2FA, it is necessary to set a little bit of terminology that will be used throughout the paper.

Given two binary numbers expressed in the following form:

$$A = \sum_{i=0}^{n-1} a_i 2^i \qquad B = \sum_{i=0}^{n-1} b_i 2^i \qquad (2)$$

their product P is given by the following expression:

$$A * B = \sum_{i=0}^{n-1} a_i 2^i * \sum_{j=0}^{n-1} b_j 2^j = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} q_{i,j} 2^{i+j} = \sum_{i=0}^{2n-1} p_i 2^i \qquad (3)$$

where $q_{i,j} = a_i b_j$.

Horizontal compression allows the fast propagation of the partial results only along one dimension (horizontal) of a multiplying array; hence a careful design of the array structure should be found, which shortens the vertical propagation (i.e. propagation on the same weight) through the cells.

To achieve this goal, the following transformation is applied to (3):

$$A * B = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} q_{i,j} 2^{i+j} + \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} q_{i,j} 2^{i+j} \qquad (4)$$

The parallel multiplier implementation presented is based on equation (4). Hence, the array is constituted by two halves, each implementing one of the nx(n/2) bit multiplications in (4), plus a final

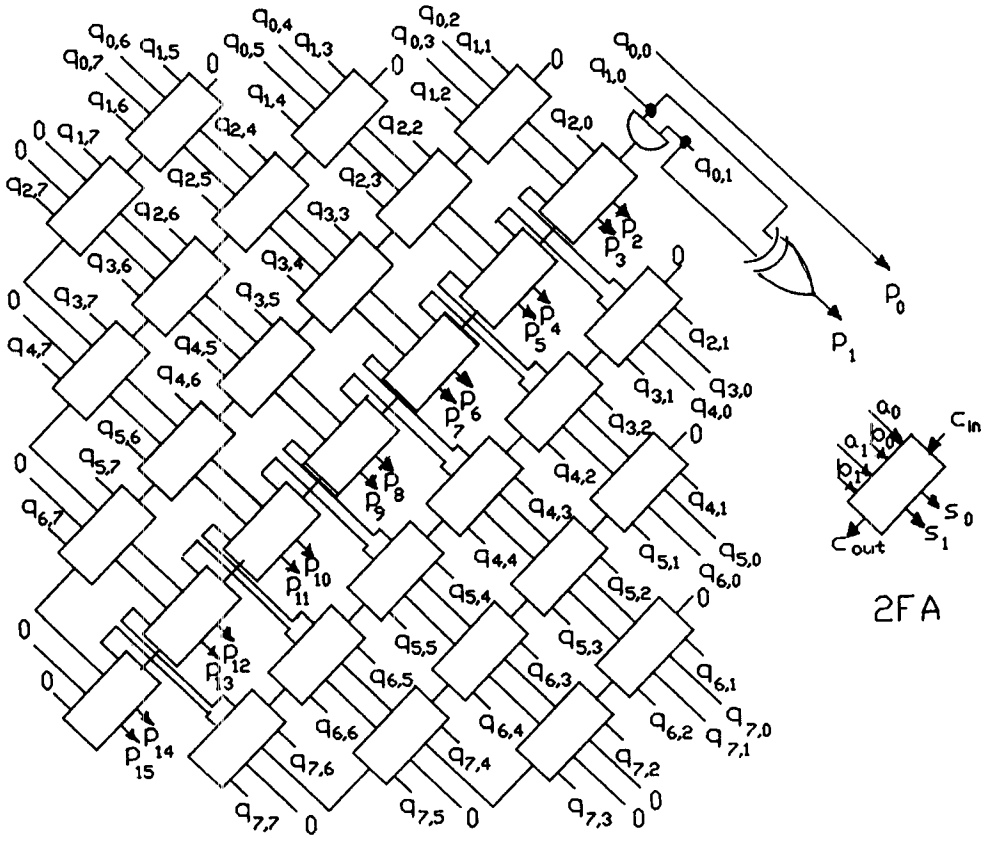

Fig. 1. 8x8 bit unsigned multiplier based on 2FAs.

horizontal adder for summing the results produced by the other two sub-arrays. Each nx(n/2) bit multiplier can be implemented so that its vertical propagation delay is roughly equal to n/2, and the final adder can also have a delay of n/2 cells, since 2FA are used that half the number of adder cells; it turns out that it is possible with this implementation to obtain an overall delay of roughly n cells for an nxn bit multiplier.

The structure of a whole 8x8 bit multiplier is shown in Fig. 1; it is possible to recognize the final adder implemented by the central row of the array. The two nx(n/2) multipliers are placed on the two opposite sides of the final adder; it is possible to note that the one below the central row is just a specular replication of the other with a shift of one position right. The same scheme as in Fig. 1 is valid for values of n being an integer multiple of 4 (or n/2 even), the structure for n/2 odd can be obtained by flipping the structure of the array in Fig. 1.

Only 2FA cells are used, except for the extra circuit required to add the least significant bit produced by the lower multiplier and the corresponding bit produced by the upper multiplier; in this case, only an half-adder is required, which is shown in Fig. 1 by separating the carry (1 AND gate) and the sum generation circuits (1 EX-OR gate).

The overall delay of a multiplier is n times the delay of a 2FA, assuming that the delay of an AND gate is shorter than that of a 2FA. A whole nxn multiplier can be built using $\left(n^2/2\right) - 1$ 2FAs and 1 half-adder.

The array of Fig. 1 is for unsigned operands, but its extension to 2's complement numbers is straightforward. According to [12] and [13], the product of a pair of 2's complement numbers is given by:

$$A * B = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \quad - b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i =$$

$$= \left(\overline{a_{n-1}}\,\overline{b_{n-1}} + 3\right) 2^{2n-2} + \sum_{i=0}^{n-2} a_{n-1}\overline{b_i}2^{n+i-1} +$$

$$+ \sum_{i=0}^{n-2} b_{n-1}\overline{a_i}2^{n+i-1} + \sum_{i=0}^{n-2}\sum_{i=0}^{n-2} a_i b_j 2^{i+j} + \left(a_{n-1} + b_{n-1}\right) 2^{n-1} \quad (5)$$

Hence the same transformation used to obtain (4) may also be applied to (5), leading to :

$$A * B = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1} q'_{i,j}2^{i+j} + \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} q'_{i,j}2^{i+j} +$$

$$\left(a_{n-1} + b_{n-1}\right)\left(2^{2n-2} + 2^{n-1}\right) \quad (6)$$

where

$$q'_{i,j} = \begin{cases} a_i\,\overline{b_j} & \text{if } i = n-1 \text{ and } j < n-1 \\ \overline{a_i}\,b_j & \text{if } i < n-1 \text{ and } j = n-1 \\ \overline{a_i}\,\overline{b_i} & \text{if } i = n-1 \text{ and } j = n-1 \\ a_i\,b_j & \text{otherwise} \end{cases} \quad (7)$$

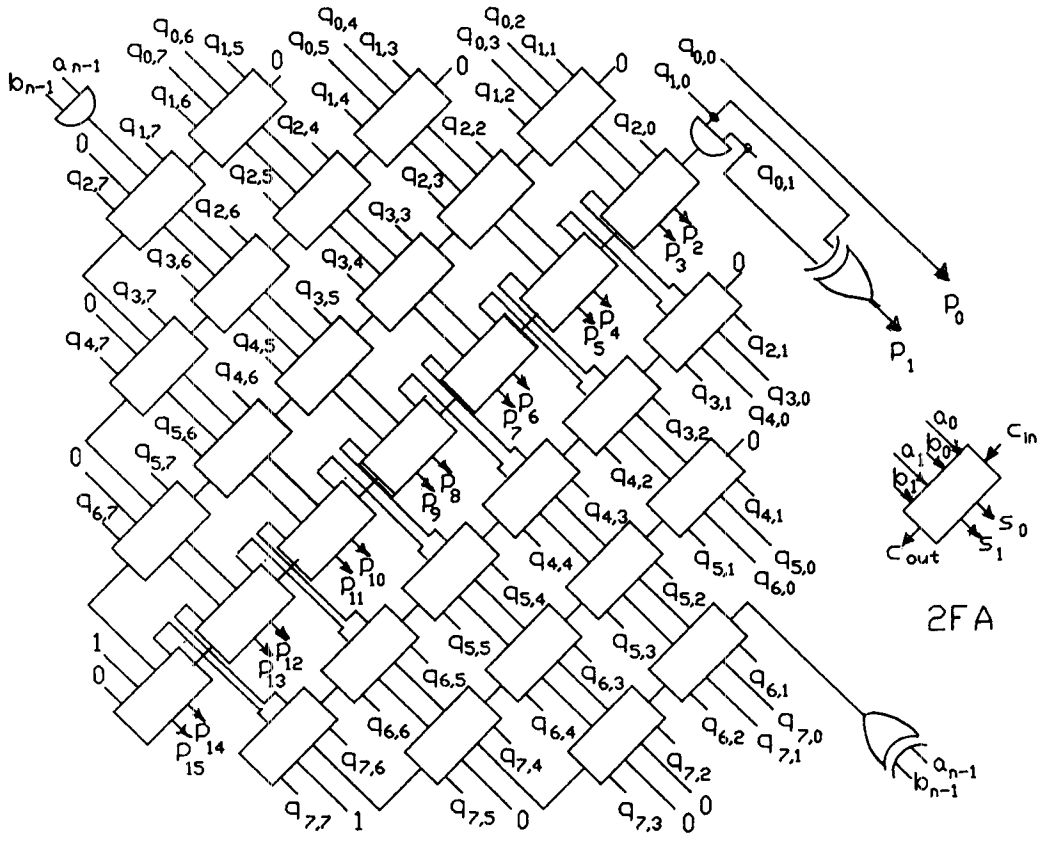

Fig. 2. 8x8 bit 2's complement multiplier based on 2FAs.

In this case, the two nx(n/2) bit multipliers have a slightly different distribution of elementary partial products to be added; neverthless, their structure is still the same as shown in Fig. 2 for an 8x8 bit multiplier. The additional half-adder shown in Fig. 2, in the form of separated sum and carry functions, is required for the addition of the term $(a_{n-1} + b_{n-1})\, 2^{n-1}$.
The delay of the array for signed multiplication is increased by only the delay of a single exclusive-or gate, respect to the multiplier for unsigned numbers. The cost is increased only by 1 half-adder, respect to the multiplier of Fig. 1.

3 Second multiplying array

The multiplier presented in section 2 achieves its high speed, by using two different techniques to speed up the vertical and horizontal propagation of the partial results. Horizontal propagation is accelerated by the particular cell used, because it is able to produce only one output carry 2 positions left of the input one, thus, if the 2FA is implemented with the same delay as the 1FA, the carry propagation speed is doubled compared to classical multipliers [11]-[12].
Viceversa, the vertical propagation of partial results does not depend directly on the cell used, because it is achieved by halving the whole multiplier and allowing the two parts to work in parallel, finally the two results are added by a fast adder circuit. It is important to note that the vertical propagation delay of each half multiplier in Fig. 2 is equal to roughly n/2 cells; the same delay could be achieved implementing the two nx(n/2) bit multipliers with 1FA.
The multiplying array presented in this section is based on the same rationale introduced in section 2. However, in this case, the use of 2FA will be limted only to the implementation of the final adder, required to sum the results produced by the two halves of the multiplier, because cells implementing a fast horizontal propagation for the final addition cannot be avoided, if a fast array is needed. The remainder of the array will be implemented by using 1FAs; since most of the array complexity is due to the implementation of the two nx(n/2) bit multipliers, the overall complexity of this second array will be smaller than for the multiplier presented in section 2, while retaining the same speed characteristics.
Though based on the same principle, this new array differs from the first one in the method used to implement the two halves of the whole array. In the multiplying structure presented in section 2, the two nx(n/2) bit multipliers are implemented using a ripple carry structure [21], since the fast horizontal propagation, guaranteed by the 2FAs, allows us to achieve an horizontal delay of $n-1$ cells, no larger than the propagation delay of the final adder. If 1FAs are used, the ripple carry structure would lead to a delay of $2n-1$ cells; hence, a carry save technique is adopted to implement the two nx(n/2) bit multipliers.
It turns out that the final adder has an increased complexity, because it should sum now the outputs of two pure carry save multipliers; in other words, the final addition involves 4 numbers (2 for each carry save multiplier) rather than only 2, as in the multiplier in section 2. The general structure of the resulting array is shown in Fig. 3, for n=8.
The final adder is implemented by 3 rows of 2FAs, organized as a small tree of ripple carry adders used to reduce the four numbers to only one. However, other solutions are possible for the final addition; one of them consists in the implementation of a cell able to perform the following arithmetic operation at once:

$$\sum_{i=0}^{3} s_i 2^i = 2\sum_{i=0}^{4} x_i + \sum_{i=0}^{4} y_i \qquad (8)$$

Using the cells defined by the equation (8), the final adder can be implemented as shown in Fig. 4; the structure of this second version of the array is closer to the structure of the array in Fig. 2. In the first version of the multiplier presented in this section, the delay of the arrays is equal to n times the single cell delay while in the second the delay is only n-1 times the cell delay.
The complexity of the two versions are different, because of the different solutions adopted for the final adder; both versions require $(n-3)^2 - 1$ 1FAs to implement the two nx(n/2) bit multipliers. The array of Fig. 3 includes $3n-5$ 2FAs, while the array in Fig. 4 needs $n-1$ cells for the final addition.

4 Discussion

This section is devoted to the evaluation of costs and performances of the solutions presented in the paper; furthermore the figures obtained for the proposed solution will be compared with those obtained for other parallel multipliers presented in the literature.
One of the problems in the assessment of costs and performances for arithmetic units is to establish the method to compute the complexity and delay of a given solution, in a general way. One method used in [9]-[14] is based on the assumption that the array cells are implemented by table look-up or ROM circuits; even when this assumption does not hold, this method is supposed to give the relative complexity of different arithmetic circuits. Using this method, the complexity of a given arithmetic unit is measured by the total number of ROM bits required to implement the unit.
Assuming a ROM implementation of the cells, it is possible to obtain the formulas, shown in Table I, for the complexity, and in Table II for the speed of the solutions presented in the previous sections (1st and 2nd multiplier), the carry-save array [11] and the two parallel multipliers presented in [14], one based on full adders and with a large final addition circuit , (6,3,4)-counter multiplier, the other based on (5,3) counters; for all the arrays considered, the complexity is evaluated in terms of ROM bits for implementing the cells plus the number of gates required to implement the extra logic.
The speed is computed in terms of cells and gate delays, because ROM-based implementation of cells leads to a delay independent of the cell size; however, both the multiplier in [14] based on (5,3) counters, and the multiplier presented in section 2 employ additional circuits. It is possible to merge the AND gates with the cell receiving their outputs, but this would increase by one the inputs, doubling the number of ROM bits.
The formulas in Table I and Table II show that the classical carry-save array is the simplest, but the slowest solution, among those considered. Viceversa, the 1st multiplier, presented in section 2, is faster and more expensive; furthermore, the cost is slightly lower than for the array based on (5,3) counters, and the delay is improved, because only 1 external AND gate, rather than n-1, contributes to the overall delay. On the other hand, the multiplier of section 2 is more expensive than both the solutions presented in section 3 and the multiplier based on (6,3,4)-counters presented in [14]; the reason for the lower cost is that the latter arrays use cells bigger than 1FA only for the final adder, with only a relatively small increase in the overall complexity, though the array is no longer uniform, since both 1FAs and addition cells are used.
The cheapest array, except for the carry-save, is represented by the 2nd multiplier with final adder implemented using 2FAs, while the most expensive is the second version of the array presented in section 3, with the final adder implemented with bigger cells; anyway, the difference in cost is not large, because they differ almost

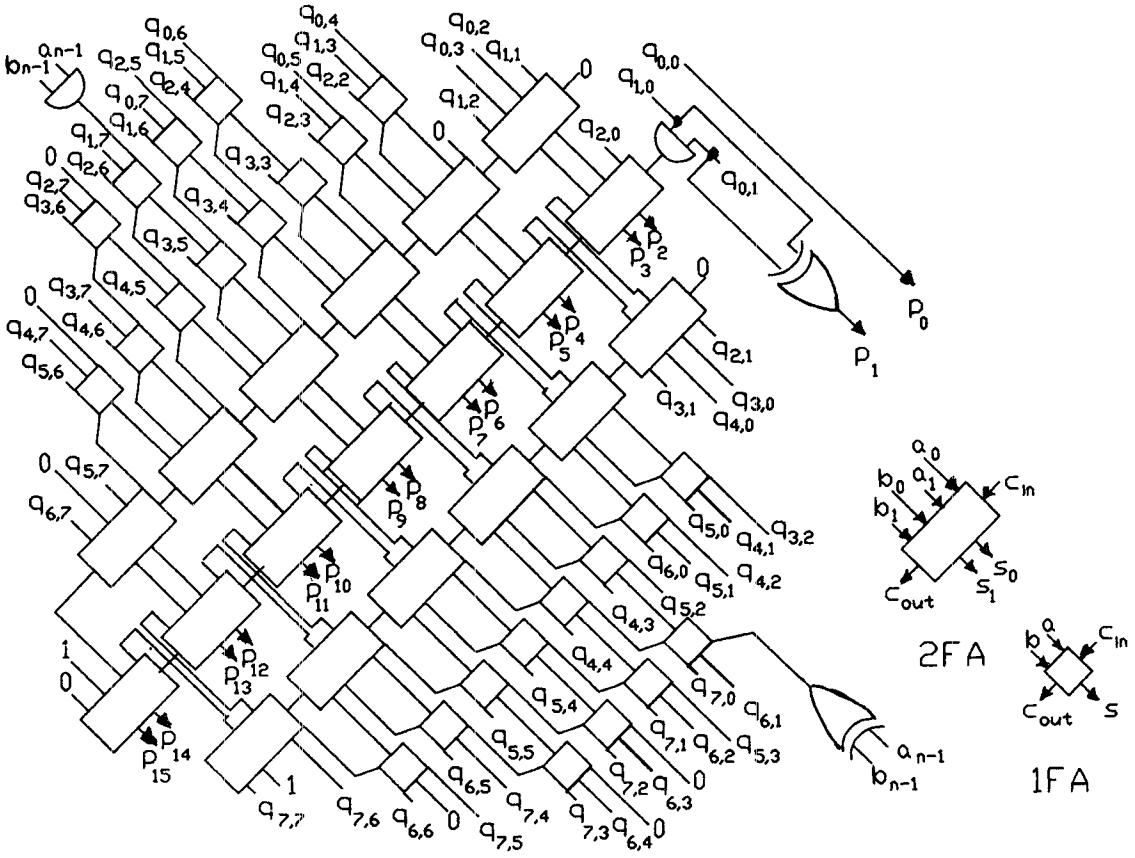

Fig. 3. 8x8 bit 2's complement multiplier based on 1FAs and 2FAs.

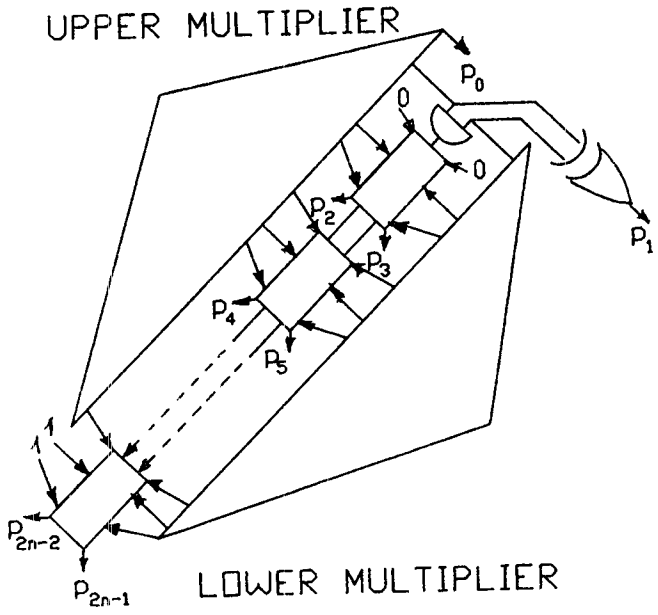

Fig. 4. Structure of the 8x8 bit 2's complement multiplier based on 1FAs and cells defined in eq. (8).

Table I
Cost breakdown for nxn bit 2's complement multipliers

Multiplier	ROM bits	EX-OR gates	AND gates	OR gates
carry-save	$16n^2 - 16n$	0	0	0
(5,3) counters [14]	$48n^2$	$n-1$	$n-1$	1
(6,3,4) counters [14]	$16n^2 + 1968n - 1952$	0	0	0
1st multiplier	$\left(48n^2/2 - 96\right)$	2	2	0
2nd multiplier (Fig. 3)	$16\,(n-3)^2 - 16 + 48\,(3n-5)$	2	2	0
2nd multiplier (Fig. 4)	$16(n-3)^2 - 16 + 4096(n-1)$	2	2	0

Table II
Delay breakdown for nxn bit 2's complement multipliers

Multiplier	Cells	EX-OR gates	AND gates	OR gates
carry-save	$2n-2$	0	0	0
(5,3) counters [14]	n	0	$n-1$	1
(6,3,4) counters [14]	n	0	0	0
1st multiplier	n	1	0	0
2nd multiplier (Fig. 3)	n	1	0	0
2nd multiplier (Fig. 4)	$n-1$	1	0	0

only in the implementation of the final adder. However, it should be considered that the solution with the final adder implemented by 2FAs can be easily implemented with other techniques than table look-up, and with a short delay; while it is unlikely that complex arithmetic functions, such as that performed by the (6,3,4) cells in [14] or the cells of equation (8), would be implemented in VLSI with the same delay as a 1FA; hence the 1st multiplier and the first versiond of the 2nd multiplier are better suited for a fast VLSI implementation using random logic.

Only 2FAs have been considered so far.as horizontal compressors; however, it is possible to extend the array construction given in section 2 to generic nFAs. The introduction of these new blocks is strictly connected with the possibility to use the equation (6) more than once during the multiplier definition; in other words, since the multipliers of section 2 and section 3 are implemented by two smaller multipliers and a final adder, it is possible to apply the same splitting procedure further on, until the size of the basic multipliers used reaches a given value.

Since each splitting of the multiplying array, as shown for the whole multiplier, leads to half the vertical propagation delay, the split implementation of the two halves of the whole multiplier will lead to a vertical propagation delay of n/4. However, if only 2FAs are used, the final addition could never be performed faster than for the arrays in sections 2 and 3, resulting in an overall delay close to 3n/4; the reason is that only vertical propagation delay has been improved. If a delay of nearly n/2 should be achieved, it is necessary that bigger horizontal compressors would be used; in particular, 4FAs should be introduced in order to speed up the final additions. It turns out that the implementation of faster multiplying circuits using this approach requires even bigger horizontal compressors.

5 Conclusion

This paper has presented several implementations of fast parallel multipliers; all the solutions have the distinctive feature that the speed increase is obtained by using only horizontal compression, which allows to obtain a fast horizontal propagation of the partial results.

Since the horizontal propagation is accelerated by using horizontal compression, the structure of the array is designed to decrease the vertical propagation delay, since both strongly affect the overall performances. Hence the vertical delay is improved by implementing an nxn bit multiplier by adding the results of two nx(n/2) bit multipliers.

Two types of multiplying arrays are obtained using this technique, both with a delay of roughly n cells. The implementation of the first solution requires only 2FAs and a little extra logic arranged according to a regular structure. For the second, two versions have been shown: the first is based on 1FAs and 2FAs and achieves a delay of n cells with a low cost, while the second has almost the same delay, a larger cost, but a more regular structure.

It has also been shown that the solutions presented compare favorably with similar multipliers presented in the literature, since they achieve better performances at lower costs; furthermore, since two of the arrays presented are based on simple arithmetic functions, they can be easily used to implement VSLI multipliers.

References

[1] E.E. Swartzlander Jr., "The quasi-serial multiplier", IEEE Trans. on Computers, vol. C-22, n. 4, Apr. 1973, pp. 317-321.

[2] I.N. Chen and R. Willoner, "An O(n) parallel multiplier with bit sequential input and output", IEEE Trans. on Computers, vol. C-28, n. 10, Oct. 1979, pp.721-727.

[3] N.R. Strader and V.T. Rhyne, "A canonical bit sequential multiplier", IEEE Trans. on Computers, vol. C-31, n. 8, Aug. 1982, pp. 791-795.

[4] R. Gnanasekaran, "On a bit-serial input bit-serial output multiplier", IEEE Trans. on Computers, vol. C-32, n. 9, Sep. 1983, pp. 878-880.

[5] L. Dadda nad D. Ferrari, "Digital multipliers a unified approach", Alta Frequenza, vol. 37, n. 11, Nov. 1968, pp. 1079-1089.

[6] R. Gnanasekaran, "A fast serail-parallel binary multiplier", IEEE Trans. on Computers, vol. C-34, n. 8, Aug. 1985, pp.741-744.

[7] C.S. Wallace, "A suggetsion for a fast multiplier", IEEE Trans. on Electronic Computers, vol. EC-13, n. 1, Feb. 1964, pp. 14-17.

[8] L. Dadda, "Some schemes for parallel multipliers", Alta Frequenza, vol. 8, n. 5, May 1965, pp. 349-356. Reprinted in: Bencjmark papers in Computer Arithmetic (E.E. Swartzlander Jr. ed.), Dowden-Hutchinson, 1980.

[9] W.J. Stenzel, W.J. Kubitz and G.H. Garcia, "A compact high-speed parallel multiplication scheme", IEEE Trans. on Computers, vol. C-26, n.10, Oct. 1977, pp. 948-957.

[10] S. Bandyopadhay, S. Basu and A.K. Choudhury, "An iterative array for multiplication of signed binary numbers", IEEE Trans. on Computers, vol. C-21, n. 8, Aug. 1972, pp. 921-922.

[11] K. Hwang, "Global and modular two's complement multipliers". IEEE Trans. on Computers, vol. C-28, n. 4, Apr. 1979, pp. 300-306.

[12] C.R. Baugh and B.A. Wooley, "A two's complement parallel array multiplication algorithm", IEEE Trans. on Computers, vol. C-22, n. 12, Dec. 1973.

[13] P.E. Blankeship, "Comments on 'A two's complement parallel array multiplication algorithm'", IEEE Trans. on Computers, vol C-23, n. 12, Dec. 1974, pag. 1327.

[14] S. Nakamura, "Algorithms for iterative array multiplication", IEEE Trans. on Computers, vol. C-35, n. 8, Aug. 1986, pp.713-719.

[15] A.R. Meo, "Arithmetic networks and their minimization using a new line of elementary units", IEEE Trans. on Computers, vol. C-24, n. 3, Mar. 1975, pp. 281-190.

[16] D.D. Gaijski, "Parallel compressors", IEEE trans. on Computers, vol. C-29, n. 5, May 1980, pp. 393-398.

[17] E.E. Swartzlander Jr., "Parallel counters", IEEE Trans. on Computers, vol. C-22, n. 11, Nov. 1973, pp. 1021-1024.

[18] L. Dadda, "Composite parallel counters", IEEE Trans. on Computers, vol. C-29, n.10, Oct. 1980, pp.942-946.

[19] L. Ciminiera and A. Serra, "Fast iterative multiplying arrays", Proc. 6th Sym. on Computer Arithmetic, Aarhus (Denmark), Jun. 1983, pp. 60-66.

[20] V. Milutinovic, D. Fura and W. Helbig, "An introduction to GaAs microprocessor architecture for VLSI", Computer, vol. 18, n. 3, Mar. 1986, pp. 30-42.

[21] H.H. Guild, "Fully iterative fast array for binary multiplication and fast addition", Electronics Letters, vol. 5, May 1969, pag. 263.