FAST ITERATIVE MULTIPLYING ARRAY

L. Ciminiera and A. Serra

CENS - Dip. di Automatica e Informatica - Politecnico di Torino
Corso Duca degli Abruzzi, 24 - 10129 Torino - Italy

Abstract

A high speed multiplying array is presented. It is based on a new cell, which is able to generate and add a rectangular block of elementary products. A careful design of the cell allows us to obtain a small delay for the signals which should be propagated through the whole array. This feature leads to a remarkable improvement in the array speed.

1. Introduction

The design of high speed multiplying circuits is one of the key points for the implementation of arithmetic units with high performances. A class of solutions presented in the literature uses an array of AND gates generating all the 1x1 bit elementary products, followed by a circuit with irregular connections, which performs the addition of the elementary products. The implementation of such additive structures, by using full-adders is discussed in [1,2]; methods for minimizing either the delay or the complexity of the non-iterative multipliers have also been presented in the literature [3,4].

Another approach uses an array composed of small basic cells almost all of the same type, interconnected following an iterative pattern. This solution is more suitable for VLSI implementation of large multipliers. On the other hand, the time required to compute the final result is $O(\log n)$ for the non-iterative multipliers and $O(n)$ for the iterative ones, where n is the number of bits of the factors.

Technological evolution has influenced the solutions proposed. When only MSI technology was available, the design of high speed multipliers was approached by introducing even larger blocks, called compressors, which, in the context of non-iterative structures, were able to perform the addition of large subsets of elementary products, thus decreasing the number of steps required to obtain the final result. Stenzel et al. proposed [5] an implementation based on m x m bit multipliers and counters larger than the (3,2) one, both implemented by means of ROM. Gaijski [6] introduces the use of parallel compressors, whose main features are: avoidance of carry propagation until the final sum, high compression capability and a small number of I/O lines; different implementations using a tree of full-adders, ROM, PLA, or smaller parallel compressors are shown in that paper. Another approach to the design of compressors is presented by Lim [7], where the internal structure of the block is composed of full-adders connected according to an ad hoc interconnection pattern.

In all the cases mentioned above, the irregular interconnections only influence the layout of a printed circuit board, hence it is not a crucial problem.

With the advent of VLSI technology, the need for a regular structure has become more evident. Indeed, a much faster, cheaper and optimized design can be achieved with iterative structures. This paper presents a new multiplying array, which may be considered an intermediate solution between the non-iterative multipliers and the iterative ones composed of small cells.

Basically, the array is still iterative, however the cells are more complex than the full-adder; the increased cell dimensions allow us to decrease the computation time by carefully designing its internal structure. For this purpose, it should be realized that the signals in the array may be subdivided into two classes, the signals influencing the computations of the other cells should go through a subset of blocks in the cell with a small total delay, while the other signals can follow slower paths through the cell. Furthermore, the final adder required by the carry-save structure is implemented by using carry look-ahead blocks, to obtain an additional speed enhancement.

In section 2, the internal block structure and the whole array is introduced. In section 3, the implementation of each cell is discussed. In section 4, the speed of the proposed array is evaluated and compared with that achieved by a classical iterative structure.

2. The array structure

Given two unsigned numbers expressed as follows:

$$A = \sum_{i=0}^{n-1} a_i 2^i \qquad B = \sum_{j=0}^{m-1} b_j 2^j \qquad (2.1)$$

their product XY is obtained by computing the following double addition:

$$AB = \left(\sum_{i=0}^{n-1} a_i 2^i\right)\left(\sum_{j=0}^{m-1} b_j 2^j\right) =$$

$$= \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} a_i b_j 2^{i+j} \qquad (2.2)$$

Thus, in order to compute the product, by means of an iterative structure, the basic cell of the array should be able to generate a subset of the elementary products $a_i b_j$ and to add them with the partial results computed by other subsets of the array. The arithmetic function performed by each cell of the array presented is shown in the dot diagram of Fig. 1. The block of dots enclosed by the solid line represents the set of the elementary products generated and added by the cell. The dots outside the solid line represent the inputs resulting from the computations of the other cells in the array. The addition is performed in two steps, as shown in Fig. 1. First, the elementary products are generated and added; then the results of the first sums are added to the other inputs of the cell.

It is worth noting that the addition of the elementary products leads to a result expressed by a pair of binary numbers, the sum of which gives the real value of the addition of the elementary products generated inside the cell.

The internal block structure of the cell is shown in Fig. 2. The COM block performs the addition of the elementary products and produces a result expressed by two binary numbers; since we constrain the sum of these numbers to be expressed by at most $2k_1$ bits, at least one of the two output numbers produced by COM can be $2k_1-1$ bit long, while the other may require $2k_1$ bits, as shown in the middle section of the dot diagram in Fig. 1.

The two sets of k_1 least significant bits produced by COM are added by means of a k_1 bit carry look-ahead adder (CLA1); while the two sets of most significant bits are added by another k_1 bit carry look-ahead circuit (CLA2), the result of this addition provides the k_1 most significant bits produced by the cell.

Since we need to express the sum of k_2

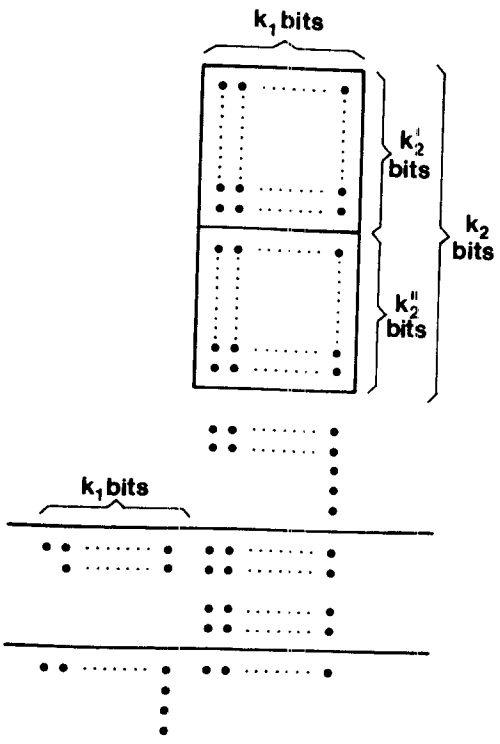

Fig. 1. Arithmetic function performed by each macrocell.macrocell.

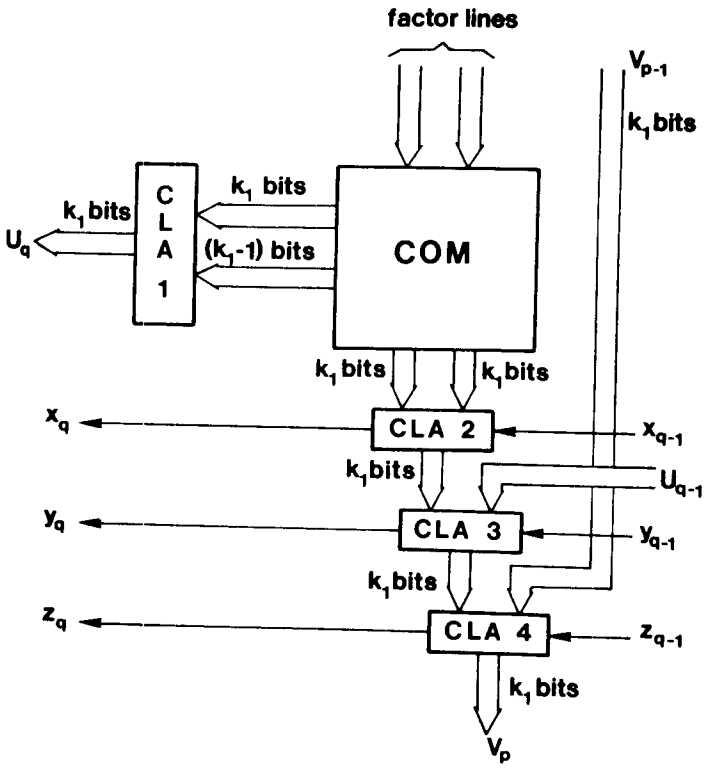

Fig. 2. Internal structure of a single cell.

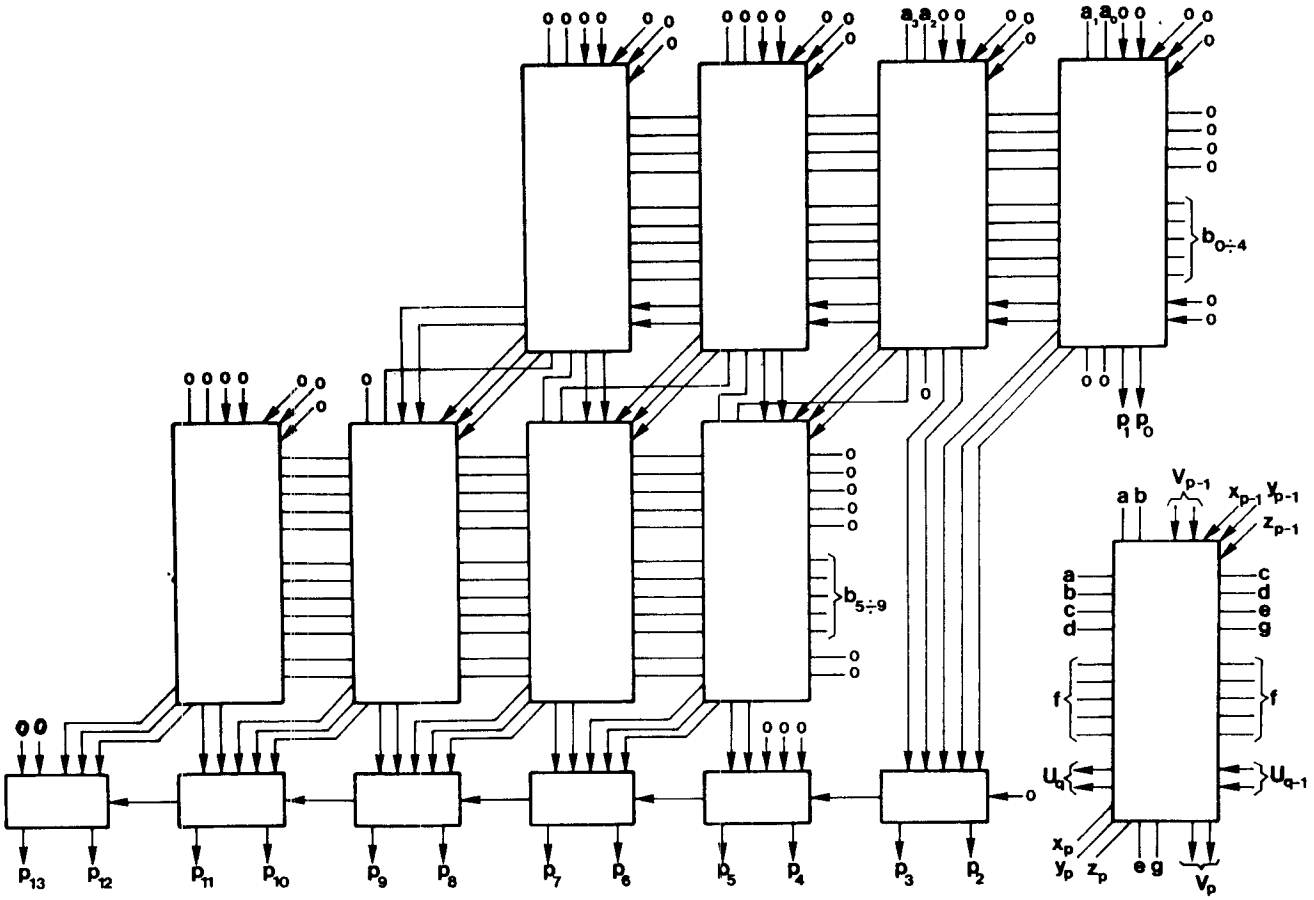

Fig. 3. 4x10 bit multiplying array.

numbers of k_1 bits by using at most $2k_1$ bits, the following relationship must hold between the values of k_1 and k_2:

$$k_2 \leq \frac{2^{2k_1} - 1}{2^{k_1} - 1} \qquad (2.3)$$

The sum of the k_1 least significant bits of the COM blocks is then added to the k_1 bits from the U_{q-1} input, by means of CLA3; the result is in turn added to the k_1 bits of the input V_{p-1}, by using CLA4.

The inputs x_{p-1}, y_{p-1} and z_{p-1} have the least significant position among the bits processed by the cell, hence they are added by using the carry-in input of CLA1, CLA3 and CLA4. Analogously, the outputs x_p, y_p and z_p have a weight 2^{k_1} times greater than x_{p-1}, y_{p-1} and z_{p-1}, as required by the dot diagram in Fig. 1.

Despite the complexity of the cell and the large number of elementary products processed, the delay introduced by one of such cells is relatively small. Indeed, the output U_p is computed after a delay from the beginning of the multiplication, which does not depend on the carry propagation through the array. With the increase in the number of bits of the operands, the increase of the operation time of the whole array depends only on the propagation delay of the signals x, y, z and V through a single cell. From Fig. 2, it can be seen that the cell has been designed so that this delay is minimized.

The array for the 4 x 10 bit multiplication is shown in Fig. 3, the cells used have $k_1 = 2$ and $k_2 = 5$. it is easy to recognize that the array is the superposition of two well-known interconnection patterns used for multiplying arrays. Indeed, the U outputs and inputs are connected in a ripple-carry like structure; since U_p does not depend on U_{p-1}, the excessive propagation delay of this interconnection pattern does not influence the speed of the array. On the other hand, the most critical signals for the array speed are connected in carry-save-like

structure, which is the fastest method to propagate the carries in a multiplying structure; however, in this case, the result produced must be reduced to a single number, by using a final adder.

The adder required by the array is composed of cells performing the arithmetic function described by the dot diagram of Fig. 4.

It is easy to verify that the result of each additive cell requires only $k_1 + 1$ bits, provided that $k_1 \geq 2$.

The additive cells may be connected in a ripple-carry structure, or it is possible to obtain a multilevel carry look-ahead structure, by using conventional look-ahead carry generators.

3. Macrocell implementation

The complexity of the cell mainly depends on the values of k_2 and k_1 and on the implementation of the COM block. The latter may be implemented by means of ROM circuits. In this case, the address signals are the factor bits which generate the elementary products added by the cell. If a result expressed by a single number is required, $2k_2 + k_1 - 1$ bits are needed to address the $2k_1$ bit ROM cell storing the result. Hence, the COM block and CLA1 and CLA2 could be implemented by means of a $2^{2k_2 - 1 + k_1} \times 2k_1$ bit ROM.

The addition of two k_1 bit carry look-ahead adders and the possibility to express the result of the COM block by using two numbers lead to a remarkable decrease in the ROM capacity. If the rectangle of elementary products is broken down into two smaller rectangles, with dimension $k_1 \times k'_2$ and $k_1 \times k''_2$, as shown in Fig. 1, the COM block may be implemented by using two ROMs connected as in Fig. 5.

In this case the total number of bits required by the two ROMs is given by:

$$2^{2k'_2 + k_1 - 1} \times 2k_1 + 2^{2k''_2 + k_1 - 1} \times (2k_1 - 1) \qquad (3.1)$$

where $k'_2 = \lceil k_2/2 \rceil$ and $k''_2 = \lfloor k_2/2 \rfloor$.

The advantage of this implementation may be shown by means of an example. Let $k_1 = 2$ and $k_2 = 5$; the implementation using a single ROM requires $2^{10 + 1} \times 4 = 8$k bits, while the implementation using 2 ROMs requires $2^{4 + 3} \times 4 + 2^{3 + 2} \times 3 = 578$ bits plus two 2 bit carry look-ahead circuits. In this case, by using two more 2 bit fast adders, the whole capacity of the ROM is reduced by one order of magnitude. Larger reduction factors are obtained for larger values of k_2.

The other basic component of a cell is the k_1 bit carry look-ahead adder. Since integration technologies, such as MOS, impose severe constraints on the fan-in, this parameter should be taken into account in order to obtain realistic estimations of the circuit complexity. In our case, the implementation shown in [8] is adopted.

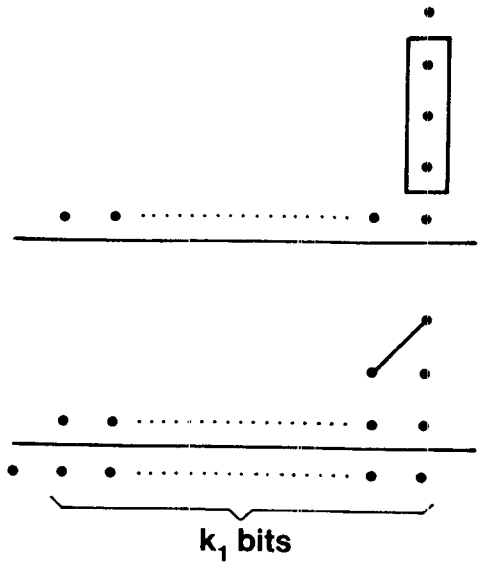

Fig. 4. Arithmetic function performed by each additive cell.

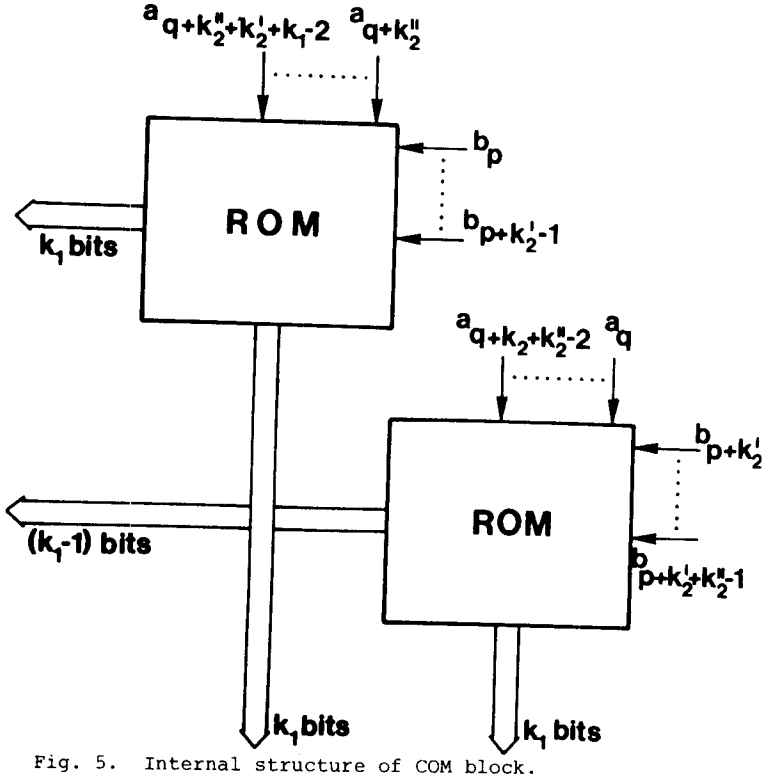

Fig. 5. Internal structure of COM block.

The maximum fan-in required is only $k_1 + 1$; from the equation (2.3) it can be seen that k_2 can be increased faster than k_1, leading to a cell adding a large number of elementary products. For example, with a fan-in 5 we can have $k_1 = 4$, which leads to a maximum value of $k_2 = 17$. The delay of the implementation adopted for the fast adders is equal to 6 times the single gate delay to produce the sum, and it is equal to 3 times the single gate delay to produce the most significant carry. Each k_1 bit carry look-ahead requires $10k_1 + 0.5\ k_1\ (k_1 + 1)$ gates. Hence the implementation of a cell with $k_1 = 2$ and $k_2 = 5$ requires 92 gates and 578 bits of ROM.

The additive cells placed at the end of the array cannot be implemented by using only two levels of gates, if we need to have a small fan-in. This problem can be solved by using the implementation shown in Fig. 6. The signals x, y and z produced by the cells in the last row are added, by means of a full-adder circuit. It is easy to verify that in this way the arithmetic function required to add the full-adder outputs, the V signals and the carry-in is a subset of the k_1 bit adder. This adder is implemented in Fig. 6 by using once again the same k_1 bit carry look-ahead circuit as in the other cells.

4. Speed evaluation

The carry-save macrocellular array (CSMA) presented is composed of two basic blocks: a k_1-bit carry-look-ahead adder, and the COM compressor block shown in Fig. 2.

For our speed analysis, it will be assumed that t_{COM} is the delay introduced by the COM block, t_c and t_s are the time needed by the CLA to produce the carry-out and the sum signals, respectively. Let 0 be the starting time of the multiplication, after a delay $t_u = t_{COM} + t_s$ the U output of every cell in the array assumes their steady-state values.

The speed analysis begins by examining the first row of cells; since there is no carry propagation through the cells within each row, all the outputs of the cells on the same row assume their steady-state values at the same time.

For the first row the following expression may be derived, taking account of the cell structure:

$$t_{x_1} = t_{COM} + t_c \qquad (4.1)$$

$$t_{y_1} = t_{COM} + t_s + t_c = t_{x_1} + t_s = t_u + t_c \qquad (4.2)$$

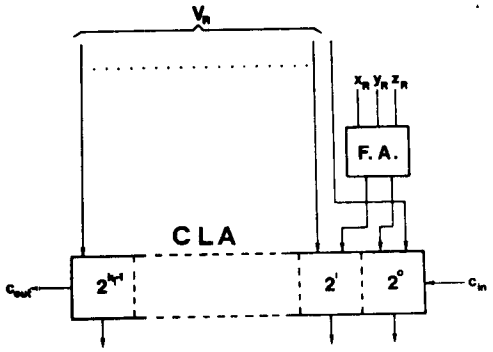

Fig. 6. Internal structure of a single additive cell.

$$t_{z_1} = t_{COM} + 2t_s + t_c = t_{y_1} + t_s \qquad (4.3)$$

$$t_{v_1} = t_{COM} + 3t_s \qquad (4.4)$$

where t_{x_1}, t_{y_1}, t_{z_1} and t_{v_1} are the instants at which the x_1, y_1, z_1 and V_1 outputs become valid.

For any row P (P 1) of the array, the time required to obtain valid outputs may be expressed as follows:

$$t_{x_p} = t_{x_{p-1}} + t_c \qquad (4.5)$$

$$t_{y_p} = t_c + \max\left\{(t_{x_{p-1}} + t_s),\ t_{y_{p-1}}\right\} = t_c + t_{y_{p-1}} = t_{x_p} + t_s \qquad (4.6)$$

$$t_{z_p} = t_c + \max\left\{t_{v_{p-1}},\ t_{z_{p-1}},\ (t_{y_{p-1}} + t_s)\right\} = t_{v_{p-1}} + t_c \qquad (4.7)$$

$$t_{v_p} = t_{v_{p-1}} + t_s \qquad (4.8)$$

The max functions appearing in the expressions of t_{y_p} and t_{z_p} are evaluated assuming $t_c < t_s$ and considering the skewing among the propagation signals introduced by the first row.

The previous equations show that the V outputs of a cell are those requiring the maximum delay; hence the computation time T_{cs} of the carry-save array, excluding the final sum, corresponds to t_{v_R}, where R is the number of rows in the structure. T_{cs} can be computed recursively by using the equations (4.4) (4.8), in this way the

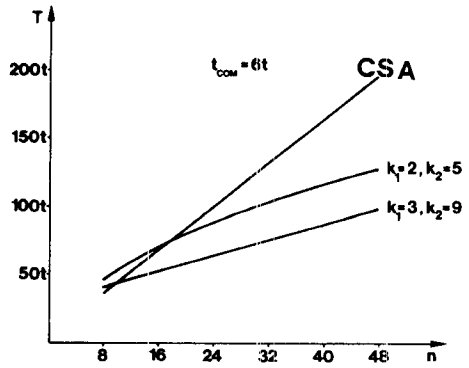

Fig. 7. Operation delay vs. the number of bits used for the factors.

following closed form expression is obtained:

$$T_{cs} = (R + 2)t_s + t_{COM} \qquad (4.9)$$

Since the number of rows in the array is given by the following formula:

$$R = \left\lceil \frac{n}{k_2} \right\rceil \qquad (4.10)$$

it follows that

$$T_{cs} = \left(\left\lceil \frac{n}{k_2} \right\rceil + 2 \right) t_s + t_{COM} \approx \frac{6n}{k_2} t \quad \text{(for large n)} \qquad (4.11)$$

With the CLA implementation considered in section 3, we obtain $t_s = 6t$ and $t_c = 3t$, where t is the single gate delay.

In order to obtain the total operation time T, it is necessary to evaluate the time needed to compute the final sum, T_A. For this purpose, it should be noted that $t_{v_R} = t_{z_R} + t_s - t_c$; since $t_{x_R} < t_{y_R} < t_{z_R}$, the delay required to obtain valid outputs values for the full-adders of the additive cell is $t_{z_R} + t_{FA}$, where t_{FA} is the delay of a full adder. This circuit may be easily implemented so that $t_{FA} = 3t$; moreover, $t_s - t_c$ is also equal to 3t, hence $t_{v_R} = t_{z_R} + t_s - t_c = t_{z_R} + t_{FA}$. Thus, both the V_R outputs and the outputs of the full-adder used in the additive cells assume their valid state at the same time T_{cs}. It follows that T_A is the time required to perform the addition of two numbers expressed by Ck_1 bits, by means of k_1 bit carry look-ahead blocks; C represents the number of cells in each row and is given by the following expression for a n x n multiplier:

$$C = \lceil (n + k_2 - 1) / k_1 \rceil \qquad (4.12)$$

Thus the value of T_A may be computed as follows:

$$T_A = C(t_c - t) + t_s - t \qquad (4.13)$$

Since $T = T_{cs} + T_A$, the total operation time may be obtained by adding the results of equations (4.13) and (4.11).

It is interesting to compare the speed of the array proposed with that of the classical carry-save array with a final adder based on k_1 bit carry look-ahead blocks. In this case the time required by the final addition, for a n x n bit multiplier, is as follows:

$$T'_A = \left(\left\lceil \frac{n}{k_1} \right\rceil - 1 \right) t_c - t + t_s - t \qquad (4.14)$$

While the delay of the carry-save array itself is given by

$$T'_{cs} = nt_{FA} = 3nt \qquad (4.15)$$

Fig. 7 shows the plot of the operation time for different values of n. It may be noted that the macrocellular structure is faster than the classical one and the speed improvement increases for large values of n.

5. Conclusions

A high speed multiplication array has been described in this paper. It is based on a new cell, which is able to generate and add a rectangular block of elementary products.

Despite of the complexity of the arithmetic function performed, a careful design of the internal structure of the cell allows us to obtain small delays for the signals which should propagate through the whole array. The same distinction between propagating and non-propagating signals is used to interconnect the cells. Indeed, propagating signals are connected following a carry-save-like pattern, to obtain a small delay, while the non-propagating outputs follow a ripple carry-like pattern, in order to simplify the final addition required by the carry-save structures.

The final result is that the time required for the final addition is roughly equal both for the classical and the macrocellular arrays, while, in the latter case, a speed up factor of 2 or more may be achieved for the carry-save array itself.

References

1. L. Dadda, "Some schemes for parallel multipliers", Alta Frequenza, vol. 19, pp. 349-356, May 1965.
2. C.S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electronic Computers, vol. EC-13, n. 2, pp. 14-17, Feb. 1964.

3. A.R. Meo, "Arithmetic Networks and Their Minimization Using a New Line of Elementary Units", IEEE Trans. Computers, vol. C-24, n. 3, pp. 281-290, March 1975.
4. R. De Mori, M. Elia and A. Serra, "Minimization method for macrocellular arithmetic networks", Proc. 3rd Symp. Computer Arithmetic, pp. 232-240, November 1975.
5. W.J. Stenzel, W.J. Kubitz and G.H. Garcia, "A compact high speed parallel multiplication scheme", IEEE Trans. Comput., vol. C-26, n. 10, pp. 948-957, October 1977.
6. D.D. Gaijski, "Parallel compressors", IEEE Trans. Computers, vol. C-29, n. 5, pp. 393-398, May 1980.
7. R.S. Lim, "High Speed Multiplication and Multiple Summand Addition", Proc. 4th Symposium on Computer Arithmetic, pp. 149-153, October 1978.
8. D.E. Atkins, S.C. Ong, "A Comparison of Two Approaches to Multi- Operand Binary Addition", Proc. 4th Symposium on Computer Arithmetic, pp. 125-139, October 1978.