ARITHMETIC ARRAY FOR FAST INNER PRODUCT EVALUATION

L. Ciminiera - A. Serra

CENS - Istituto di Elettrotecnica Generale, Politecnico di Torino
Corso Duca degli Abruzzi, 24 - 10129 TORINO - Italy

Abstract

The paper presents a new fast arithmetic array, suitable for VLSI implementation, which computes the inner product of two vectors. The operands and the result are expressed using the 2's complement notation, which is the most general and flexible one. Cells performing the 2x2 bit full multiplication are used for obtaining a reduction of the operation time.
A particular pipelining scheme, with different degrees of latching, is used in order to implement parallel computations with a moderate cost increase. Graphs showing the characteristics and the advantage domain of the proposed array are presented. An IC implementation of the proposed array could have a speed from 5 to 10 times greater than the multiplier-accumulator circuits currently available.

1. Introduction

The evaluation of the inner product of two vectors is one of the most important arithmetic computations in the field of digital signal processing. In fact, both non-recursive digital filters and correlators require an arithmetic processor performing the inner product. Since the speed of such processors affects mainly the bandwidth of the signals which may be processed, a great deal of attention has been devoted to the study of efficient hardware implementations of arithmetic circuits performing signal correlation and non-recursive digital filtering. In particular, the works of De Mori[2] and Peled and Liu[5] regarding non-recursive digital filters should be mentioned.
Actually, some integrated circuits performing the multiplication-addition function are available on the market[1]; on the other and, the rapid development of the large scale (LSI) and very large scale integration (VLSI) technologies will enable to put in a single chip arithmetic circuits faster and more complex than those actually available.
In this paper, a new arithmetic array, performing the inner product of two vectors, is presented. Such an array is composed of a signed multiplier and an adder circuit, so that the whole computation is performed recursively. The array accepts operands expressed using the 2's complement notation and produces the result using the same representation. The 2's complement multiplication is performed using an algorithm, presented here, which is derived from the Baugh and Wooley algorithm[7].
In order to achieve an high speed implementation of this signed multiplication algorithm, the same approach, successfully used for pipelined unsigned multipliers[3], is applied. The basic cell (macrocell) used for implementing the array performs the 2x2 bit rather than the 1x1 bit full multiplication.
Combining the use of macrocells and the pipeline technique, the implementations presented in this paper achieve a very high speed, without a drastic cost increase, because of the particular pipelining scheme adopted. This fact allows one put the array presented here in a single VLSI chip.
The evaluation of the array performance in based on the total cost G_T, the operation time τ and a global parameter η, called efficiency and defined as follows:

$$\eta = \frac{1}{G_T \tau} \qquad (1)$$

A comparison with some arrays previously presented in the literature shows that the macrocellular array is suitable for implementing inner product processors having an operating speed not achievable using other implementations.
In section 2, the 2's complement multiplication algorithm and the internal structure of the macrocells are presented. In section 3, the pipelining scheme is described and formulas for evaluating the arrays characteristics are derived. In section 4, the arrays presented here are compared with two analogous arrays previously presented in the literature.

2. Basic array

The computations required for evaluating the inner product of two vectors M elements long are defined by the following formula:

$$S = \sum_{p=1}^{M} X_p Y_p \qquad (2)$$

The equation (2) may be computed using the following recursive procedure:

$$S_p = S_{p-1} + X_p Y_p \qquad p=1,\ldots,M \qquad (3)$$

with

$$S_o = 0 \quad \text{and} \quad S = S_M \,. \qquad (4)$$

The arrays presented in this paper are able to perform the calculation required by one step of this procedure, so that the final result may be iteratively computed, applying to the inputs M pairs of operands at different time intervals. Therefore, the arrays must compute one signed multiplication and one addition with the previously accumulated result.

In this paper, both the operands and the partial results are represented using the symmetric 2's complement notation, because this is the most flexible and widely used number representation in digital systems. Moreover, it assumed that both the operands are represented using n bits, while the partial and final results are expressed using 2n-1 bits.

Let $(x_o, x_1, \ldots, x_{n-1})$ and $(y_o, y_1, \ldots, y_{n-1})$ be the 2's complement representation of two factors X_p and Y_p, respectively, the following relations hold:

$$X_p = -x_o + \sum_{i=1}^{n-1} x_i 2^{-i} = -xo + X^* \qquad (5.a)$$

$$Y_p = -y_o + \sum_{i-1}^{n-1} y_i 2^{-i} = -y_o + Y^* \qquad (5.b)$$

where x_o and y_o are the sign bits (o for positive numbers and 1 for negative ones) and n-1 bits are used for representing X^* and Y^*. The sign bit is considered negative, so that the signed 2's complement representation gives the actual value of the number. Using this representation, the product of two numbers is given by the following formula:

$$X_p Y_p = (-x_o + X^*)\,(-y_o + Y^*) = x_o y_o - y_o X^* - x_o Y^* + X^* Y^* \qquad (6)$$

From (6) it may be deduced that the product of two signed numbers is obtained by multiplying two unsigned numbers X^* and Y^*, then the result is corrected using an additive term C, defined as follows:

$$C = x_o y_o - x_o Y^* - y_o X^* = x_o y_o + C_1 \qquad (7)$$

Since the 2's complement notation is used, the following two equations also hold:

$$-Y^* = -2^o + \sum_{i=1}^{n-1} \bar{y}_i 2^{-i} + 2^{-n+1} \qquad (8.a)$$

$$-X^* = -2^o + \sum_{i=1}^{n-1} \bar{x}_i 2^{-i} + 2^{-n+1} \qquad (8.b)$$

Hence the corrective term C_1 defined in (7), is expressed by:

$$C_1 = -(x_o + y_o)2^o + \sum_{i=1}^{n-1} x_o \bar{y}_i 2^{-i} + \sum_{i=1}^{n-1} y_o \bar{x}_i 2^{-i} + \\ + (x_o + y_o)\, 2^{-n+1} \qquad (9)$$

Since the magnitude of the factors and of the result is less than 1, we are not interested in evaluating the bits having a weight greater than 2^o, therefore the result of the signed multiplication is given by the following formula:

$$XY = X^* Y^* + \bar{x}_o y_o + (x_o \oplus y_o)\, 2^o + \sum_{i=1}^{n-1} x_o \bar{y}_i 2^{-i} + \\ + \sum_{i=1}^{n-1} y_o \bar{x}_i 2^{-i} + x_o 2^{-n+1} + y_o 2^{-n+1} \qquad (10)$$

The matrix of the elementary products, defined by (10), is shown in Fig. 1, for n=6.

The basic block of the multiplying subset of the inner-product arrays presented in this paper is a cell (macrocell) bigger than a gated full-adder. The macrocell performs the 2x2 bit full multiplication. With this choice, the number of cells in the array

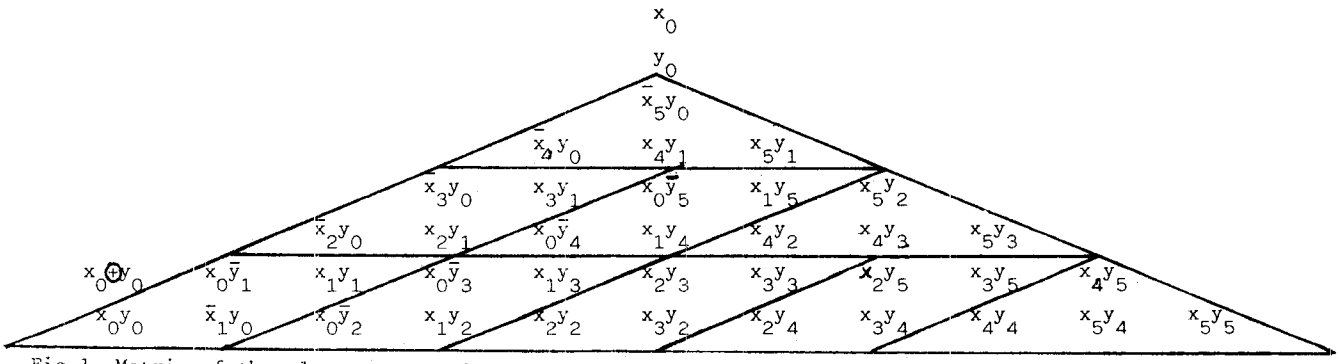

Fig.1. Matrix of the elementary products generated by the 2's complement multiplication algorithm.

decreases and the operation speed increases, if the macrocell and the gated full adder have the same delay. On the other hand, the complexity of a macrocellular multiplying array is greater than that of an array using gated full adders.

The basic block of a cellular multiplier is a mxm full multiplier, that is a cell performing AB+C++D, where A,B,C and D are numbers expressed using m bits (m < n).

A straightforward implementation of a 2x2 bit full multiplier, using two gate levels, leads to an expensive circuit, because the most significant bits of the result produced depend on a large number of input variables. The macrocell implementation used in this paper is represented by the diagram shown in Fig. 2, where a cross represents an elementary product of the matrix in Fig.1, and a dot represents an additive input or output. In Fig. 2 the macrocell logic symbol is also shown. It may be noted that the macrocell is partitioned into two independent arithmetic circuits: the first one processes the least significant addends; the second one processes the other addends. The carry generated from the first circuit in not passed to the second one but is fed to a successive macrocell, using an appropriate output.

The matrix of elementary products, shown in Fig.1, is partitionned in several blocks; the blocks are chosen so that only two pairs of factor bits appear in each block. However, it may be noted that there is no uniform rule for generating the elementary products appearing in each subset.

In fact, there are some blocks, where the factor bits always appear uncomplemented, and there are some other blocks, where some factor bit appears complemented. Since each block of elementary products must be generated by a macrocell, three types of macrocells must be used to perform the signed multiplication.

The arithmetic functions defining the first type (M1) are:

$$w_{1,2}\,2^1+w_o\,2^o = (v_o+u_o+x_{i-1}y_{j-1})\,2^o \qquad (11.a)$$

$$w_3\,2^3+w_2\,2^2+w_{1,1}\,2^1=x_iy_j2^2+(x_{i-1}\,y_j+x_iy_{j-1}+$$

$$+v_{1,1}+v_{1,2}+u_1)\,2^1 \qquad (11.b)$$

for i=2,4,..., n-2 and j=2,4,..., n-2.

The arithmetic functions defining the second type (M2) are.

$$w_{1,2}\,2^1+w_o\,2^o=(x_1y_{i-1}+u_o+v_o)2^o \qquad (12.a)$$

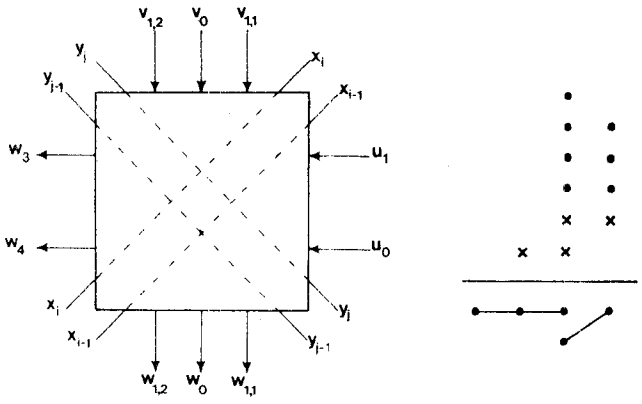

Fig.2. Logic symbol and diagram of the arithmetic function performed by each macrocell.

$$w_3\,2^3+w_2\,2^2+w_{1,1}2^1=x_o\bar{y}_j\,2^2+(x_o\bar{y}_{j-1}+x_1y_j+u_1+$$

$$+v_{1,1}+v_{1,2})2^1 \qquad (12.b)$$

with j=2,4,..., n-2; using this same cell and changing the roles of the input factors, it is possible to generate the blocks, where the x_i (i=1,..., n-1) bits appear complemented.
The third type of macrocell (M3) is defined by the following pair of equations:

$$w_{1,2}\,2^1+w_o2^o=(x_1y_1+u_o+v_o)\,2^o \qquad (13.a)$$

$$w_3 2^3+w_2\,2^2+w_{1,1}\,2^1=x_oy_o2^2+(x_o\bar{y}_1+\bar{x}_1y_o+u_1+v_{1,1}+$$

$$+v_{1,2})2^1 \qquad (13.b)$$

In Fig.3 the whole array in shown, the signed multiplication is performed by the sub-array outside of the dashed line. It should be noted that the result of the signed multiplication is represented using a redundant code. In fact, two bits are generated for every weight 2^{2t+1} (t=0,1,...) times greater than the weight of the least significant bit of the product.
In order to obtain the result represented using the 2's complement notation, the bits having the same weight must be added. Since an addition is also required by the equation (3), the array is provided with a final row of additive cells (A), performing the sum of the previously accumulated result (expressed using the 2's complement notation), and the result calculated by the multiplying sub-array (expressed using the redundant notation previously described). The additive row is shown in Fig. 3, enclosed by the dashed line. Fig. 4 shows the logical symbol of the A cells and the dot diagram, describing the arithmetic function performed.

3. Pipelined array characteristics

The pipeline technique is a useful tool for increasing the operating speed of a system, without a drastic cost increase. In particular, this technique gives better results, when the flow of data and the operations performed in each section of the system are deterministic, as happens in the arithmetic circuits. Hence, in this section, several arrays are presented; they are derived from the array shown in Fig. 3 using different degrees of pipelining, that is, the pipelined arrays considered have a different number of cell levels (K) in each

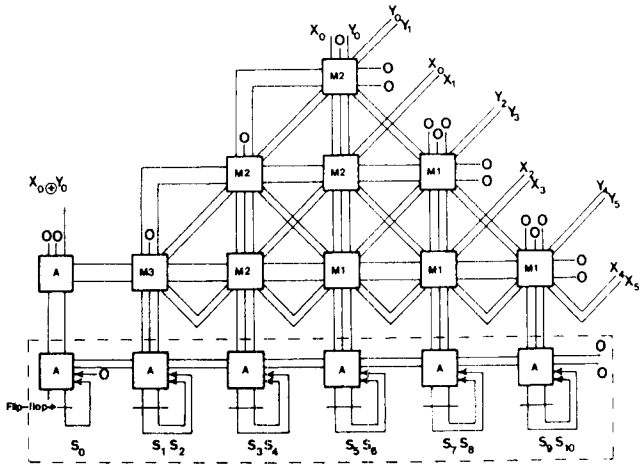

Fig.3. Unpipelined macrocellular array.

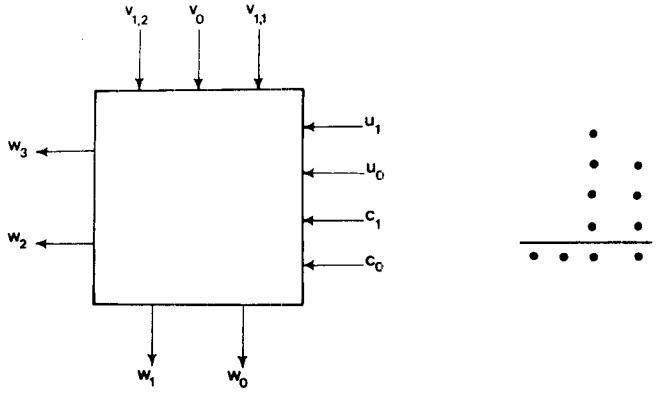

Fig.4. Logic symbol and dot diagram of the arithmetic function performed by each additive cell.

stage in the pipe.
Fig.5 and Fig.6 show the Pipelined Macrocellular Arrays (PMA), having K=1 and K=2, respectively.
It is worth noting that in both PMAs, shown in Fig. 5 and Fig.6, the memory elements required by the pipelining are placed only on the additive connections.Therefore, the last row of the multiplier does not produce the product of a pair of factors for each clock period. However, the final result is correct. In fact, we are interested in computing only the final result, which is obtained summing M matrices of elementary products, like that shown in Fig. 1, disregarding the order of this summation. In the PMAs considered here, when a new pair of factors enters into the array, the whole matrix of elementary products is generated and it is added to the bits, stored in the memory

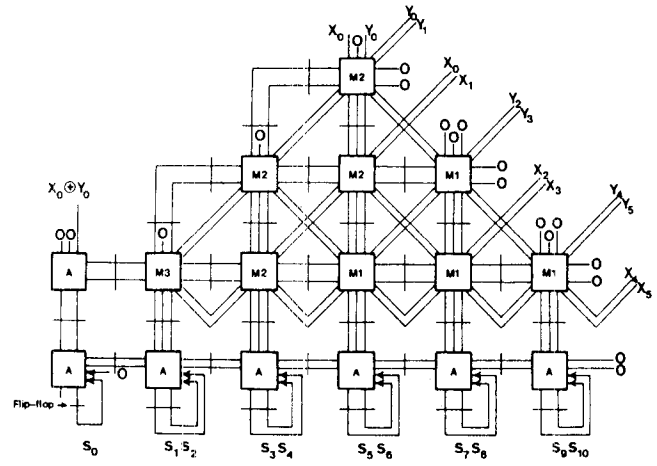

Fig.5. Maximally pipelined macrocellular array (K=1).

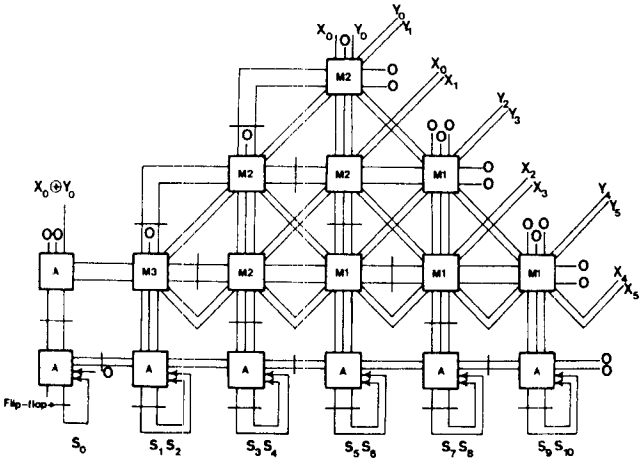

Fig.6. Pipelined macrocellular array having K=2.

elements, which have been obtained by the previously generated matrices. After M clock periods, N-1 pairs of zero factors (N is the number of stages in the pipe) are introduced into the array, so that the bits stored in the memory elements, required by the pipelining, are added to the accumulated number in order to obtain the correct result. Using this latching scheme, a considerable latch saving may be achieved.

The cost of a pipelined arithmetic array is obtained summing the cost of the unpipelined array, G_c, and the cost of the memory elements required for pipelining it, G_l. Since the implementations, using two gate levels, of the M1, M2, M3 and A macrocells require 82, 83, 81 and 95 gates, respectively, the number of gates required for implementing the pure combinational array, considered in this paper, is given by the following formula:

$$G_c = 82\,(n_r-1)^2 + 2(n_r-1)\,81 + 83 + 2n_r \cdot 95 \qquad (14)$$

where $n_r = \lceil n/2 \rceil$.

The number of gates required by the memory elements is dependent on the implementation chosen. In many previous works about pipelined arithmetic arrays[8-9], the Earle latch[6] has been considered for pipelining an arithmetic circuit. This latch may be implemented using 4 gates, on two gate levels. The number of latches embedded in the array depends on the value of K; the following formula gives the value of the number of latches in a PMA for any value of K:

$$N_l(K) = (2n-1) + \sum_{d=1}^{N-1} f(d \cdot K) \qquad (15)$$

where $N = \lceil (2n_r)/K \rceil + 1$ and the function f is defined as follows:

$$f(b) = \begin{cases} 5n_r & b = 1 \\ f(1)-1 & b = 2 \\ f(2)-2 & b = 3 \\ f(b-2)-5 & b > 3 \end{cases} \qquad (16)$$

Finally, the total number of gates required for implementing a PMA, for a given value of K, is:

$$G_T(K) = G_c + 4 \cdot N_l(K) \qquad (17)$$

At the beginning of this section, it has been outlined that the calculation of the inner product of two vectors, M elements long, using the arrays presented here, requires M clock periods, during which the M pair of the vectors elements enter into the array, plus N-1 clock periods required in order to empty the pipe. Note that, given the particular latching scheme adopted, the first result appears on the outputs of the multiplying subarray before the N-th clock period, as it happens in almost all the pipelined systems. However each of such results is not equal to any S_i. Since each clock period in a pipelined system cannot exceed the maximum delay of a stage in the pipe, D, the total time required for performing the calculations defined by (1) is:

$$\tau(M,K) = \lceil M+N(K)-1 \rceil \cdot D(K) \qquad (18)$$

The dependence of N on K has been previously shown. The delay of a stage is K times the macrocell delay plus the delay of the memory element placed at the end of the stage. Assuming the single gate delay as the time unit, the following equation gives the value of τ for any value of K:

$$\tau(M,K)=\lceil M+N(K)-1 \rceil \cdot (2K+2) \qquad (19)$$

Finally, the efficiency, of the arrays, η, may be calculated, once the values of τ and G_T have been computed, using the equations presented in this paper.

4. Array comparison

In this section, the formulas presented in the section 3 are used to compare the characteristics of the arrays presented here with some results previously presented in the literature.

Fig. 7 shows the diagram of the total cost G_T versus the number of bits used for representing the vector elements. In this figure the curves for the PMAs having K=1,2, and for the implementation proposed in[2] are drawn. It may be seen that the arrays proposed here are cheaper than the De Mori array. Furthermore, by using (14), it is possible to show that the difference between G_T, for K=1, and G_c is small; this fact shows that the pipelined scheme presented in section 3 leads only to a moderately increased cost compared with the unpipelined array.

Athough the cost of a PMA is smaller than that of the De Mori array, the former is faster; in fact, looking at Fig. 8, where the values of τ are shown, for n=32, it may be noted that the PMA obtained for K=1 has a better operation time for all the pratical values of M. As would be expected, the PMAs have higher values of the η; Fig. 9 shows the curves of η versus M for n=32.

For K=1, PMA is more efficient than De Mori array for all the pratical values of M; whereas, for K=2

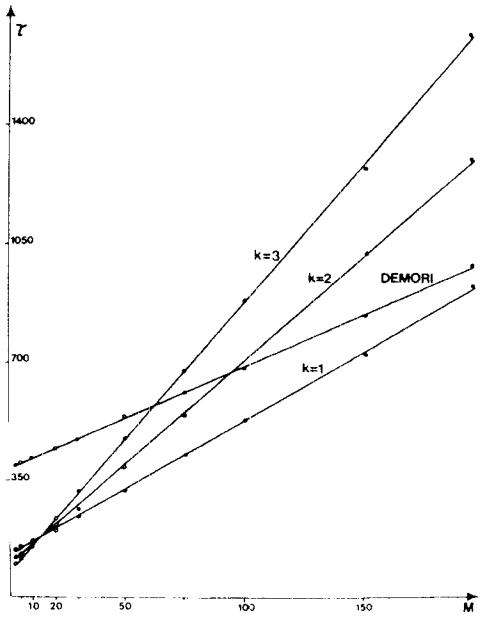

Fig.8. Operation time vs. the length of the two vectors.

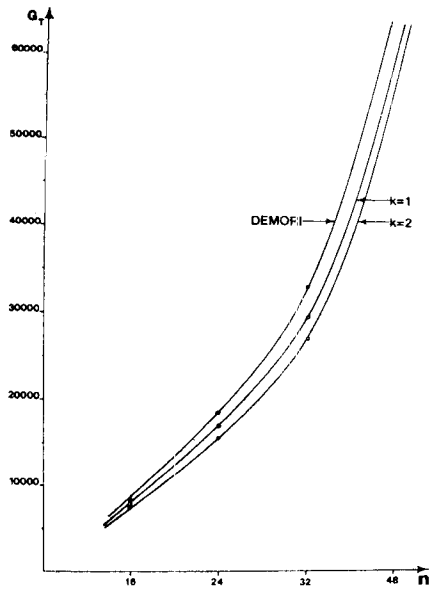

Fig.7. Total cost vs. the number of bits used for representing the operands

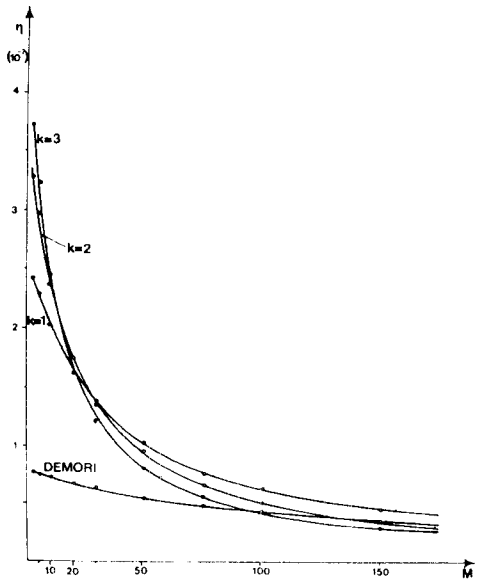

Fig.9. Efficiency vs. the length of the two vectors.

and K=3, the PMAs give large advantages for M<100, and they show small disvantages for M > 100. Another implementation, based on gated full adders, of a correlator array (GFA) has been presented[4]. The formulas giving the array characteristics are:

$$G_c = 11\,n^2 + 18\,n + 3 \qquad (20)$$

$$N_l = 2\,n\,N_a + 2\,(N_a - 1) - 0.5\,K N_a\,(N_a - 1) \qquad (21)$$

$$N_a = \left\lceil \frac{\overline{2n-1}}{k} \right\rceil + 1 \qquad (22)$$

$$\tau = (M + N_a - 1)\,(2K+2) \qquad (23)$$

It may be observed, from (23) and (19), that the PMAs allow one to implement arrays faster than the corresponding GFAs.The percentual time saving decreases when the value of M increases; hence, the PMAs seem to be more suitable than GFAs for non recursive digital filtering, which requires some tens of multiplications, rather than for signal correlation, which may require some hundred of multiplications.
Fig. 10 shows the operation time τ versus K of the PMAs and GFAs, for n=32 and M=20. The PMAs perform the inner product at a speed from 25% to 30% higher than the corresponding GFAs. On the other hand, the PMAs are more complex than the GFAs, as shown by Fig. 11.

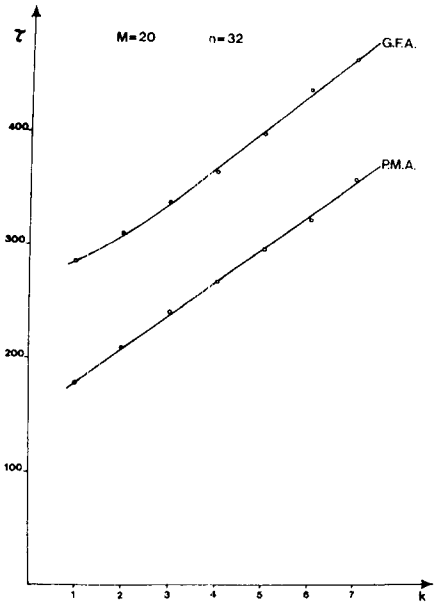

Fig.10. Operation time vs. the pipelining degree for GFAs and MPAs.

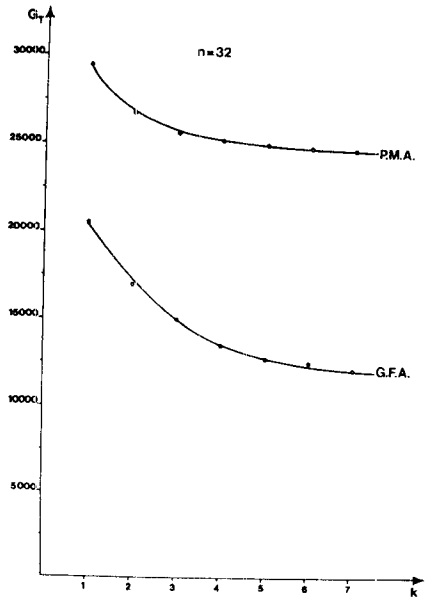

Fig.11. Total cost vs. the pipelining degree, for GFAs and PMAs.

The curves of the efficiency versus the pipelining degree are drawn in Fig. 12, for n=32 and M=20. Both curves show a maximum value, the PMA reaches the maximum efficiency for K=2 and the GFA for K=4; these values are the optimum degrees of pipelining for the two implementations, respectively.
However, if some constraint is imposed on the operation speed, the optimal implementation cannot be derived from Fig. 12. For example, if $\tau \leq 300$ is needed, from Fig. 10 it may be seen that only K=1

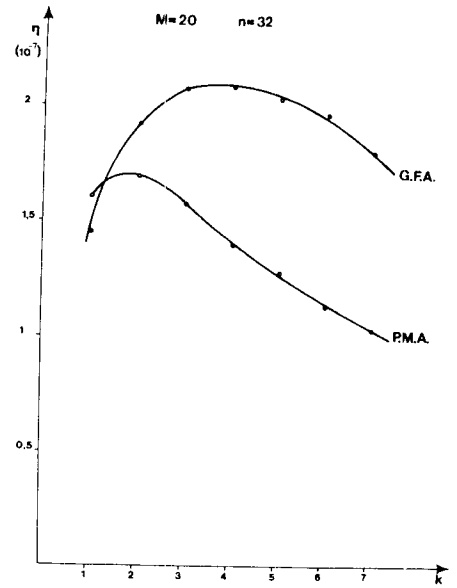

Fig.12. Efficiency vs. the pipelining degree for GFAs and PMAs.

leads to an admissible implementation, using GFAs, while the admissible set of implementation, using PMAs, includes the values of K from 1 to 5. Therefore, the optimal GFA implementation requires K=1 and the optimal PMA implementation requires K=2; from Fig. 12 it may be seen that the optimal PMA implementation is more efficient than the optimal GFA implementation. On the other hand, the PMA array is cheap enough to be implemented in a single VLSI circuit.

5. Conclusion

In this paper there is presented a new arithmetic array, which performs iteratively the inner product of two vectors. Since it accepts operands expressed in 2's complement and produces the result using the same representation, the array may easily be used in common digital systems for performing non-recursive digital filtering or signal correlation. The particular algorithm used for the signed multiplication leads to a simple and regular structure of the array. In order to achieve an high computation speed, the pipeline technique and cells biggerthan the gated full-adder are used.
Although the final result should be expressed using the 2's complement notation,the intermediate results need not be representated using the same notation. Hence, the macrocell implementation takes advantage of a particular redundant number representation in order to have a moderate complexity. Moreover a particular pipelining scheme allows us to save a relevant number of latches, so that the resulting array is cheap enough to be implemented using a single VLSI circuit.
Using formulas presented in section 3, it has been shown that the solutions presented are particularly suitable for non-recursive digital filtering and for the applications having hard real time requirements, in this case, the arrays proposed here can cover a range of operating speeds, unattainable using other implementations previously presented in the literature.

References

1. TRW LSI product "The 16 bits multiplier-accumulator, model TDC 1010J", TRW 1979.
2. De Mori R. "Cellular structures for implementing recursive and non recursive digital filters", the Radio and Electronic Engineer, vol. 46, 4 (1976), pp.175-181.
3. Ciminiera L. and Serra A. "Pipelined multipliers implemented with macrocells", Proc. of 1980 IEEE Symp. on Circuits and Systems, April 1980, pp. 393-398.
4. L. Ciminiera and A. Serra. "High speed correlator circuits" Signal Processing: Theories and Applications (M. Kunt and F. de Coulon eds.), North Holland 1980, pp. 483-489.
5. Peled A. and Liu B. "A new hardware realization of digital filters" IEEE Trans. on Acoustics, Speech and Signal Processing, vol. ASSP - 22, 6 (1974), pp. 456-462.
6. Earle, J.C., "Latched carry-save adder", IBM Tech. Disclosure Bull., 7 (1965), pp. 909-910.
7. Baugh C.R. and Wooley B.A., "A two's complement parallel array multiplication algorithm"; IEEE Trans. on Computers, vol. C-22, 12 (1973), pp. 1045-1047.
8. Jump J.R. and Ahuja S.R. "Effective pipelingng of digital systems" IEEE Trans. on Computers, vol. C-27, 9 (1978), pp. 855-865.
9. Deverell J., "Pipeline interative arithmetic arrays", IEEE Trans. on Computers, vol. C-24, 3 (1975), pp. 317-322.