SOME SCHEMES FOR FAST SERIAL INPUT MULTIPLIERS

Luigi Dadda

Dipartimento di Elettronica
Politecnico di Milano
Italy

Index terms: fast digital multipliers, parallel counters, fast computer arithmetic.

ABSTRACT

The design of fast multipliers for binary numbers represented in serial form is considered, according to a general scheme composed by an array generator and a summator. The bits of the product are generated with the least delay with respect to the operators bits. The array generator computes the elements of the multiplier array. The summator computes the sum of the array elements in order to generate the product bits. The array elements can be generated according to two different general schemes: the first computes all the new array elements at each step (arranged on a diagonal and on a row of the multiplier array), the second computes the multiplier array elements column by column. Several schemes of array generators are given and compared, and for each of them a suitable summator using parallel counters is illustrated.

1. Introduction

In a growing number of applications it becomes important to implement fast digital multipliers: digital signal processors, correlators, image processors, etc. represent examples of such applications. Fast digital multipliers assume different forms, depending also on how the operands and the product are represented. Operands represented as binary numbers whose bits are simultaneously available, i.e. as numbers in parallel form, require multipliers of the type proposed by Wallace /1/ and Dadda /2/ which produce the product also in parallel form. This class of multipliers has been extensively studied /5,9,10/. In some cases, one of the operands is available as a binary parallel number, while the other is in serial form, i.e. its bits are available in succession, usually with the least significant bit as first. Multipliers suitable for such operands have been proposed by Dadda and Ferrari /4/ and Swartzlander /6/. In other cases, both operands are represented as binary serial numbers, both bits of the same weight appearing simultaneously (the least significant bit first). Fast multipliers for such cases have been proposed by Dadda and Ferrari /4/ and more recently by Chen and Willoner /12/. Trivedi and Ercegovac /8/ have developed algorithms for division and multiplication of number in serial form with the most significant bit as first. The purpose of this paper is to present a class of serial multipliers, which includes schemes already proposed and offers new schemes. Serial multipliers are well suited for the VLSI implementation of complex algorithms (for instance, for signal processing) involving a large number of interconnected processor performing multiply/add operation. In comparison with a fully parallel multiplier, a serial multiplier is slower but it requires less silicon area due to its simplest structure and to the use of serial rather then parallel interconnections.

2. General consideration on multipliers with serial binary inputs

Let us assume that the two operands are represented as serial binary natural, whole numbers, whose bits of the same weight appear simultaneously, the least significant bits being the first. The operation of the fastest possible multiplier can be represented as in fig. 1 example (for operands of n=4 bits). Operands bits of the same weights appear at the multipliers inputs in time slots t_0, t_1, t_2, t_3. The product bit p_0 (the least significant bit of P) can be computed immediately after a_0 and b_0 have been applied: p_1 can be computed as soon as a_1, b_1 have appeared, etc. Immediately after the application of the most significant bits, a_4 and b_4, all the remaining bits of the products can be computed. In other words, the least significant half of P of the fastest possible multiplier for serial numbers is also a serial number, while the most significant half of P can, in principle, be available as a parallel binary number. In a variation of the ideal, serial inputs multiplier, also the most significant part of P could be obtained in serial form, following the least significant half of P at the same output.

The schemes which will be illustrated, are based on a common structure, composed of two cascaded parts*

- the array generator, receives the two input numbers and produces the various terms of the array M as they become available and in a form suitable for the following part:
- the summator, whose inputs are the outputs of the array generator, which produces the product P. The

* It must be noted that the proposed partition of a multiplier in two cascaded part is helpfull in finding various interesting multipliers, as it will be shown. An approach in which such a partition is not used i.e. in which the multiplier is synthesized as a whole, is given by Atrubin /3/.

array generator and the summator can be designed independently, provided the array generator outputs are in accordance with the input form required by the summator. The schemes which will be illustrated can be classified according to two different types of array generator as already done in /4/.

The first type uses the "subarray-wise" generation of the array M, as illustrated in fig. 2. When a new pair of operand bits is presented, one "row" and one "diagonal" are added to the subarray whose elements have been previously generated. Assume S_{j-1} be the value of the (j-1)th subarray generated by the j-1 least significant bits of A and B. When a_j and b_j are presented, let us call R_j the new row generated by a_j and D_j the new diagonal generated by b_j (including the term $a_j b_j$). R_j and D_j can be expressed as:

$$R_j = a_j \sum_{i=0}^{j-1} b_i \, 2^i$$

$$D_j = b_j \sum_{k=0}^{j} a_k \, 2^k$$

The value of S_j of the j-th subarray will then be:

$$S_0 = a_0 b_0$$

$$S_j = S_{j-1} + 2^j (R_j + D_j); \quad (1 \leq j \leq n-1)$$

According to this scheme, the array generator will generate R_j and D_j in parallel form and the summator will computes S_j and will generate the product P.

The second type derives from a different way of obtaining the product: instead of considering P as the result of adding the new row R_j and diagonals D_j to the value S_{j-1} of the preceding subarray, one could obtain the product P by adding "columns-wise" the numbers represented by the n rows. In other words, the array generator can be assumed to generate simultaneously the n bits of the same weight (i.e. belonging to the same column) of the rows. The role of the summator will then be to sum the n serial numbers representing the rows, and to generate the product bits.

In all schemes illustrated in this paper use will be made of (p;q) parallel counters, i.e. combinational circuits giving at their q outputs (considered as representing a binary integer) the number of the outputs ($p<2^q$) having value "one": fig. 3 represents examples of parallel counters, using a graphic notation in which the p inputs are separated from the q outputs by a horizontal line. The counters of fig. 2a,b,c,d have inputs of the same weight ("simple" counter). The concept of parallel counter can be extended to include devices capable of providing the counting of inputs having different weights: see the examples of fig. 2e,f. The following schemes will use network of counters, represented according to the above rules. For more details on parallel counters, see /13/.

3. Sub-array multiplier schemes

Multiplier schemes based on the generation of rows R_j and diagonals D_j, and on the evaluation of the sum S_j of the succeeding sub-array shall now be considered. The generation of R_j and D_j can be obtained in several forms. The conceptually simplest form (α), is obtained as in fig. 4, where for each column (i.e. for each weight) two outputs are provided, one for R_j's bits, the other for D_j's bits. The circuit is composed by two registers, for containing the multiplicand, B, and the multiplier, A, respectively. These are "stack" registers, whose pointer can be provided by an auxiliary shift register, C, with the initial content as shown. The outputs, C_j identify the subsequent time slots, and are used to store the bits a_j, b_j in subsequent positions in registers A and B. An array of AND gates generates the various elements of the array M: the outputs of such gates are ORed in each column, providing at the output pins the subsequent rows R_j and diagonals D_j as illustrated in fig. 5. D_j and R_j bits are generated, at 2n-1 and 2n-3 outputs respectively of the array generator, with two outputs for each column of the multiplier array (except for the first and the last columns, where a single output in D_j is needed).

A second method (β) of generating R_j and D_j is implemented as in fig. 6a scheme, which is composed by two shift registers and two linear arrays of AND gates, generating D_j and R_j at n and n-1 outputs, respectively.

The main difference between this circuit and the preceding one is that the weights of all outputs must be multiplied by four at each step (compare fig. 6b with fig. 5). This can be easily obtained in the summator, as it will be seen.

A third method (γ) for generating D_j and R_j is given in fig. 7a: this circuit is composed with two stack registers and two linear arrays of AND gates, generating D_j and R_j at n and n-1 outputs respectively. Note that the (n-1) stack register for the multiplicand B, is preceded by a single stage shift register in order to single-out the b_j lastly introduced. For the same purpose, a single bit register at the input of the A register is used.

It can be seen from fig. 7b that the output weights must be multiplied by two at each step. As in the previous case, this can be obtained in the summator. For each of the preceding array generators, various summators can be designed. In the simplest scheme, the sub-array sum is obtained in the form of a single binary number, which is stored in a register: a three-input parallel adder obtains the sum S_j of D_j, R_j and S_{j-1}, which is fed-back to the register. Fig. 8 represents the three schemes of summators based on the above principle, and suitable for the three types of array generators just described (note that in fig. 8 and following a small square is used to represent the binary memory elements of the register S_{j-1}). In all the above schemes, the three numbers D_j, R_j and S_{j-1} are first transformed into a set of two equivalent numbers, by means of an array of (3;2) and (2;2) parallel counters. These two numbers are then added in a two-input parallel adder (in which use can be made of carry-look-ahead circuitry, to enhance speed). The same figure shows for each scheme how the product bits, p_i, are generated. Note that the schemes in fig. 8a and b generate the least significant half's bits, p_0 to p_3 in series (at the corresponding time t_0 to t_3), while the most significant part (t_4 to t_7) is generated in parallel (at different output in fig. 8a;

at the same output pins in fig. 8b). In fig. 8c scheme, all product bits appear serially at a single output. Since in all the above schemes a parallel adder is required, the speed is essentially limited by the carry propagation.

If a greater speed is desired, one of the following schemes can be used, where carry propagation is totally avoided by representing S_j with a set of at least two equivalent binary numbers,

For each of the three array generators, fig. 9 and fig. 10 give schemes where S_j is represented by two, respectively by three, equivalent numbers.

The operation of the above circuits can be studied as shown in fig. 11 example, which refers to fig. 9c, where the significant bits in the succeeding steps are shown as dots, while points are used to represent those bits that are not affected (and remain at the initial zero value). The following remarks that can be made on the operation of such a circuit, apply also to some of the other circuits:

- from t_4 to t_7, D_j and R_j bits are all zeros, and S_j is repeatedly fed-back in order to generate in succession the bits of the most significant half of the product that could be obtained in parallel by adding in a parallel adder, S_3' and S_3'' (at time t_3).
- The bits necessary to represent S_j' and S_j'' as shown in the figure, have been found to be the only ones necessary for the purpose by means of the following procedure: first an indefinite number of bits have assumed both for S_j' and S_j'' then the operation of the circuit has been studied as done in fig. 11, determining those bits that are used as significant.
- The left most half adder of the first reduction stage is shown as producing only its least significant output, since its inputs can never be simultaneously "one": it could be therefore replaced by a two-input OR gate.
- Fig. 9c shows a scheme for n=4 operand bits. It can be shown that, for a larger number of bits, the two left most columns and the right most columns remain as in fig. 9c, and the new schemes differ only in the number of the "central" columns. Therefore, increasing by one the number of operands bits implies a circuit using two more full adders.
- The reduction of a set of four numbers to an equivalent set of two numbers can be obtained in a single stage, using (4,4;4) counters: see /13/ for more details.
- Some of the bits shown as significant in fig. 11 (i.e. represented by a dot) are always zero (pseudo-significant") at some steps: this is the consequence of using a redundant representation for S_j, or of the use of non-saturated counters with dont-care combinations.

More remarks can be done by examining the operation of the remaining circuits using the above method. Schemes using type γ array generators (fig. 8c, 9c, 10c) are the only ones producing the product P in serial form from a given output, while in the remaining circuits the succeeding product bits appear at different outputs. Since a register whose cells are connected to such outputs is filled in a stack-wise fashion, those outputs have been called "stack-type". Using a three numbers for S_j implies a number of memory elements larger than for the two-number solution, but requires a single stage of (4;3) and (5;3) counters, while in the two-number schemes two cascaded stages of (3;2) counters are needed. The choice between the two alternatives depends from consideration of speed and cost, which in turn depends on the technology actually adopted. Table I summarize the main parameters and features of the above circuits.

4. "Column-wise" Multiplier Schemes

A second class of serial input multipliers shall now be considered, based on the generation of the multiplier-array column by column. As already said in paragraph 2, the product can then be considered the result of adding n serial binary numbers, each represented by a row in the multiplier-array, all bits of the same weight (i.e. belonging to a column) appearing simultaneously.
The problem of generating the succeeding rows can be solved using the scheme of fig. 12: it is composed by a shift register and a stack register, whose outputs feed n AND gates.
The circuit can be derived by the one proposed by Swartzlander/6/ for a serial-parallel multiplier by replacing a shift register with a stack register.

The problem of adding the n serial numbers produced by the preceding scheme can be solved in several ways, using parallel counters and memory elements. The problem has been extensively discussed elsewhere by the author /11/. Fig. 13 shows three kind of schemes: several more can be found in the article cited. In fig. 13a a parallel counter with feed-back carries is used. The least significant bit of the counter represents the product bit; the second least significant bit is delayed by a single step before being counted with the succeeding column; the third most significant bit is delayed by two steps and counted with the second column following the present one. It can be easily shown that, if a (r;s) counter is used, columns of up to (2^s-1) bits can be handled, i.e. factors of up to $n_{max}=(2^s-1)$ bits can be multiplied (e.g. for s=3, $n_{max}=5$; for s=4, $n_{max}=12$, etc.).
In fig. 13b, the first stage consists of a parallel counter having as inputs the column bits: the binary number, output of such counter, is added (using a parallel adder) to a number composed by the values of the second, third and fourth bits of the sum obtained in the preceding step; the least significant bit of this parallel addition represents the product bit p_j for the present step t_j. A variation of the same scheme is shown in fig. 13c, which uses a carry saving technique, in order to increase the speed. A second scheme for generating the multiplier array columns is presented in fig. 14: it uses two shift registers feeding 2n-1 AND gates. As shown in the figure these gates generate two columns at a time. More precisely, in the first N steps (N=(n-1/2) if n is odd, N=n/2-1 if n is even), no columns are generated (all outputs are zero). At step N+1, the term a_0b_0 is generated (column 0). At step N+2, the terms a_0b_1, b_1a_0 (column 1) and terms a_2b_0, a_1b_1, a_0b_2 (column 2) are generated. At step N+n the last two columns will be produced. It can be said that the rows of the multipliers array are generated as serial/parallel binary

numbers. In order to obtain the product, two approaches can be taken.
With the first approach a single adder for serial number, of the same types as those illustrated in the previous case is used: using suitable gates, in the first half of each step the first column C' is applied to the counter, while the second column generated in the same step is applied to the same in the second half of the step. In other words, the two columns C', C'', generated in parallel, are serialized by means of a clock whose frequency is twice the frequency of the clock applied to the shift registers in the columns generator.

Using a second approach, the addition of the serial/parallel rows is performed by a suitable counter, which must be capable of handling two columns at a time, taking into account that all bits of the columns C'' are weighted twice the bits of the first column, C'. Moreover all circuits based on this principle work at the same clock rate used in the shift registers of the column generator. Various schemes for such adder can be designed, by extending the method given in /11/. One such scheme is presented in fig.15a for the case n=15: the first stage is composed by one (15;4) counter and one (14;4) counters, each counting the number of ones in C' and C'' respectively. The two output numbers of such counters, (each composed by four bits) are associated to two three-bits numbers, which represent the carries of the preceding two columns.
The second stage reduces the above four numbers to an equivalent set of three numbers, using full adders as shown: note that the least significant output bit of the full adder, having as inputs the three rightmost bits, represents one of the products bits, p'_j, i.e. the product bits corresponding to the C' column at clock time t_j.
The third stage reduces the above three numbers to a set of two equivalent numbers: as in the preceding stage, at least significant bit of the rightmost full adder is the product bit p''_j corresponding to the column C'' at time t_j. Besides p'_j and p''_j, the outputs thus obtained must be accounted for by feeding them back as carries for the addition to the following columns C'_{j+1} and C''_{j+1}. As C'_{j+1} and C''_{j+1} have weights four times the weights of the respective preceding columns C'_j and C''_j, these carries must be shifted two places to the right, as shown in the figure. Fig.15b shows a second example of how the problem of generating the product bits p'_j and p''_j can be solved. The circuit is similar to the carry feedback circuit used in fig. 13a which is here extended to handle two columns at a time. It can be seen from fig. 15 that, using (15;4) and (14;4) counters, n can be at most 12. The column C'_j is associated with three carries bits and counted in a (15;4) counter, whose least significant bit represents the product bit p'_j corresponding to column C'_j. The remaining three outputs are fed back and are placed in the column corresponding to their respective weights. Note is particular that the output weighted 2 is associated to column C''_j, along with other two carries, and counted in a (14;4) counter. The least significant output bit of this counter represents the product bit p''_j, while the remaining three outputs are fed-back as carries. The carries from both counters (five in total) must be delayed by a clock period before being associated with the subsequent couples of new columns. A variation of fig. 14 scheme for generating columns is represented in fig. 16: it comprises two shift registers, where the operands A and B are fed serially. While in fig. 14 schemes the operands bits are synchronous, in this new scheme they are fed and shifted alternatively, by means of two clocks, t' and t'', t'' being delayed with respect to t' by half of the common clock period. A single parallel counters of the same type of fig. 13a generates the product bits. In comparison to fig. 14 scheme, fig. 16 scheme generates the product in serial form, one bit at a time, with clock frequency which is twice the clock used for shifting the registers. No product bit will be generated for the first n/2 clock times for n even (or (n/2-1) for n odd). The first column (a_0b_0) will be generated:

-for n even, at the t' clock following the (n/2)th clock
-for n odd, at the t'' clock following the (n/2-1)th clock.

All the (n-1) columns will be generated before the two factors have been completely shifted out of their register. More precisely, the last columns will be generated:

-at the (n+1)-th t' clock time, for n even;
-at the (n+1)-th t'' clock time, for n odd.

Note, finally, that the speed of operation of the single counter used in fig. 16 is twice the speed for the composite double column counter used in fig. 15 (for the same clock frequency).
The comparison between "column-wise" and "sub-array" schemes is immediate for what concerns the array generators: besides the two input registers, n AND gates are required for the fig. 12 and 16 schemes and 2n-1 AND gates for the fig. 14 scheme.
The comparison between the summator can be done only by specifying first the solution chosen for implementing the parallel counters. If it is assumed, for example, to use full adders for the purpose, knowing that (see /7,13/) at most $N-1- \log_2 N$ of them are needed to implement an N input parallel counter, it can be shown that the fig.13a schemes uses less full adders that any of the "sub-array" schemes.
On the other hand, these have larger delays (except in the cases where pipe-lined counter can be used).

5. Timing Considerations

All multiplier schemes considered consist of an array generator feeding a summator. Assuming that all registers are clocked in synchronism, the minimum clock interval Tcl must be: Tcl < Td + Tsett where:
Tsett is the setting time of the registers;
Td is the longest delay in the combinatorial parts, comprising two parts: Td = Tag + Tsu

where: Tag is the delay of the output gates in the array generator (AND-OR in fig.4, AND in fig. 6,7,12, 14,16) Tsu is the delay of the combinatorial part of the summator.

In conclusion:

$$Tcl < Tsett + Tag + Tsu$$

The various schemes illustrated so far differ mainly in Tsu, which can be determined in relation to the structure chosen for the summator. The following remarks can be made with reference to the various

schemes.
. In fig. 8 schemes

$$Tsu = Tfa + Tad(n)$$

where: Tfa is the full adder delay
Tad(n) is the delay in the adder (a carry-look-ahead will of course be used for minimum Tad(n) which is a function of n, the number of stages.

. For fig. 9 schemes: $Tsu = 2Tfa$
. For fig. 10 schemes: $Tsu = T(5;3)$

where: T(5;3) is the delay of a (5;3) counter.

It can be shown that a (5;3) counter can be composed by 3 full adder, with a delay of 3Tfa. It can also be obtained by means of a 2 level AND-OR network. The use of multiplier based on column generation can be treated similarly.
In fig. 12 and 13 schemes:

$$Tcl < Tag + Tco$$

where: Tco is the delay of the parallel counter in fig. 13a; or: the sum of the parallel counter delay and of a full adder delay in fig. 13b or of two full adder delays in fig. 13c.

In order to increase the speed of operation, pipe-line techniques can be used. It seems not possible to apply them at cases where partial sums (as in the case of sub-array multipliers) or carries (as in fig. 13a) are fed back. A pipe-line parallel counter can be used in fig. 13b scheme. A pipe-line parallel counter can be easily obtained by composition of smaller (elementary) parallel counters by inserting memory elements between the cascaded stages. If full adders are assumed as elementary counters, then it will be:

$$Tcl < Tsett + Tfa$$

which is the minimum achievable with a given technology. A pipe-line introduces also a latency time Tl that is Tcl times the number of the stages of the parallel counter. The reader is referred to /13/ for more details.

6. Conclusion

It has been shown how binary digital multipliers for factors represented in serial form can be designed, according to different principles. Two main classes of serial multipliers have been considered. The first is based on the fact that, whenever a new bit for both factors is presented, a new row and a new diagonal of the multiplier-array becomes available. The product is obtained by accumulating these rows and diagonals in the subsequent steps. The second class considers the product as the result of adding $\underline{n}$ serial binary numbers, corresponding to the rows of the multiplier-array. At each step, all bits belonging to a column of the multiplier-array are determined and then summed to obtained the products. It has also been shown that the various schemes thus obtained are based on the use of shift registers and stack registers, and that the product can be obtained serially bit by bit from a single output or from different outputs. In some of the circuits shown, the most significant half of product can be obtained also in parallel form. In all circuits parallel counters as basic building blocks are used. A comparison between the various schemes shows that some of them offer definite advantages as far as the number of gates and registers needed is concerned. A fully significant comparison could be made by considering the various ways of implementing parallel counters, the technology adopted with the related problem of routing and of silicon area evaluation. This requires further investigation for the selection of the optimum multiplier for a given application. The schemes described refer to the case of operands having the same lenght: they can be easily extended to the case of operands with unequal lenght which for sake of brevity has not been treated explicitely.

REFERENCES

/1/ C.S.Wallace: A suggestion for a fast multiplier, IEEE Trans. Electronic Computer, vol. EC-13, pp. 14-17, Feb. 1964.

/2/ L.Dadda: Some schemes for parallel multipliers, Alta Frequenza, vol. 34, n. 5, pp. 349-356, May 1965.

/3/ A.J.Atrubin: A one-dimensional real-time iterative multiplier, IEEE Trans. Electronic Computers, vol. EC-14, pp. 394-399, June 1965.

/4/ L.Dadda, D.Ferrari: Digital multipliers a unified approach, Alta Frequenza, vol. 37, n. 11, pp. 1079-1089, Nov. 1968.

/5/ A.Habibi, P.A.Wintz: Fast multipliers, IEEE Trans. Comput., vol. C-19, pp. 153-157, Feb. 1970.

/6/ E.E.Swartzlander, Jr.: The quasi serial multiplier, IEEE Trans. Comput., vol. C-22, pp.317-321, April 1973.

/7/ E.E.Swartzlander, Jr.: Parallel counters, IEEE Trans. Comput., vol. C-22, pp. 1021-1024, Nov. 1973.

/8/ K.Trivedi, M.Ercegovac: On-line algorithms for division and multiplication, IEEE Trans. Comput. vol. C-26, pp. 681-687, July 1977.

/9/ A.Stenzel, B.Kubitz, C.Garcia, A compact high-speed parallel multiplication scheme, IEEE Trans. Comput., vol. C-26, pp. 948-957, Oct. 1977.

/10/ L.Dadda: On parallel digital multipliers, Alta Frequenza, vol. 40, pp. 574-580, Oct. 1978.

/11/ L.Dadda: Multiple addition of binary serial numbers, IEEE 4th Symp. Computer Arithmetics, pp. 140-148, Santa Monica, Oct. 1978.

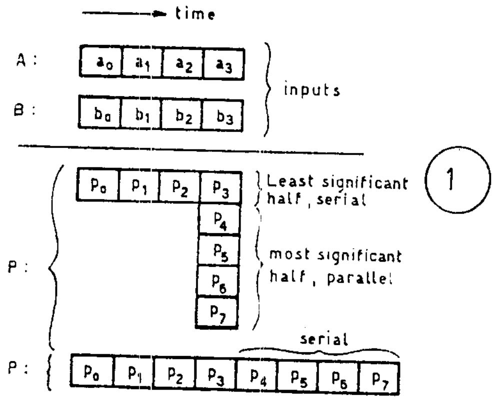
/12/ I.N.Chen, R.Willoner: An O(n) parallel multiplier with bit sequential input and output, IEEE Trans. Computers, vol. C-28, n. 10, pp. 721-727, Oct. 1979.

/13/ L.Dadda: Composite parallel counters, IEEE Trans. Comput., vol. C-29, n. 10, pp. 942-946, Oct. 1980.

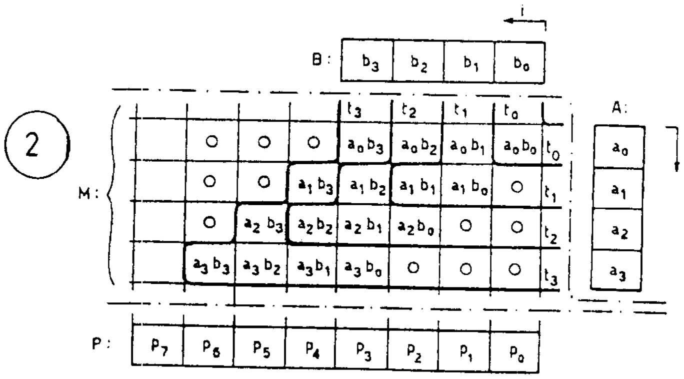
TABLE I : Characteristics of multipliers based on array generators by diagonals and rows

* Parallel, with additional circuits

types:	α: array generator outputs with fixed weights			β: array generator outputs to be multiplied by 4 at each step			γ: array generator outputs to be multiplied by 2 at each step		
input registers:	stack			shift			stack, modified		
array generator outputs:	4n-4			2n-1			2n-1		
array gen. AND gates:	n^2			2n-1			2n-1		
OR gates:	2(n-2)			--			--		
input to AND gates:	$3n^2 - 2n+1$			4n-2			4n-2		
array gen. fig. n.:	4,5			6			7		
summator fig. n.:	8a	9a	10a	8b	9b	10b	8c	9c	10c
rows for S_j	1	2	3	1	2	3	1	2	3
(2;2) counters	2	4	1	--	3	n+1	--	1	--
(3;3) counters	2n-3	4n-7	1	n-2	2n-3	--	n-1	2n-3	--
(4;3) counters	--	--	3	--	--	--	--	--	1
(5;3) counters	--	--	2n-6	--	--	n-2	--	--	n-2
parallel adders stages	2n-1	--	--	n-1	--	--	n-1	--	--
carries p_j spilled	2n	4n-3	6n-7	n-1	2n+2	4n-4	n-1	2n-3	3n-4
p_j stored	2n	4n-3	6n-7	2(n-1)	3n+1	6n-5	--	--	--
outputs: stack sing.half	stack	stack	stack	stack	stack	stack	serial	serial	serial
most sign.half	parallel	stack*	stack*	parallel	stack*	stack*	serial/ par.	serial*	serial*

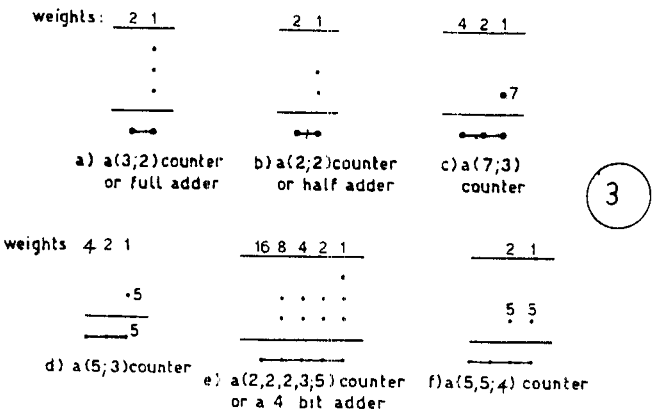


1

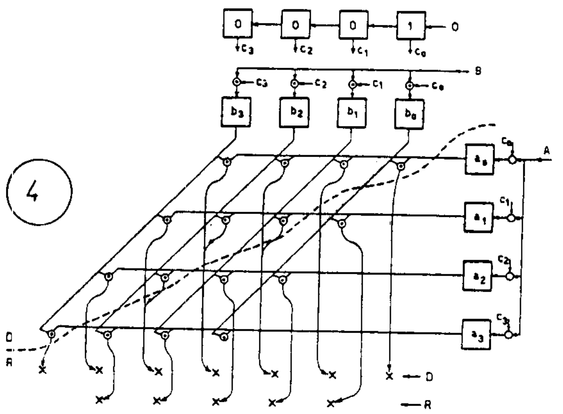


B : b_3 b_2 b_1 b_0
t_3 t_2 t_1 t_0
M :
a_0b_3 a_0b_2 a_0b_1 a_0b_0
a_1b_3 a_1b_2 a_1b_1 a_1b_0
a_2b_3 a_2b_2 a_2b_1 a_2b_0
a_3b_3 a_3b_2 a_3b_1 a_3b_0
A: a_0 a_1 a_2 a_3
k
P : p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

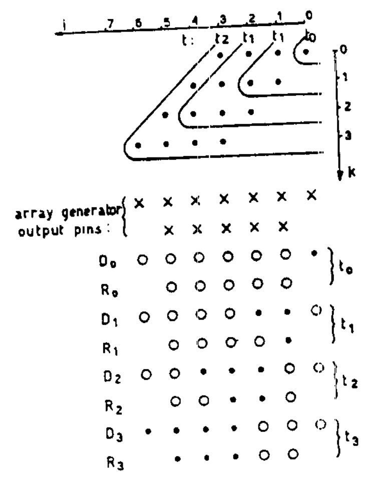
2



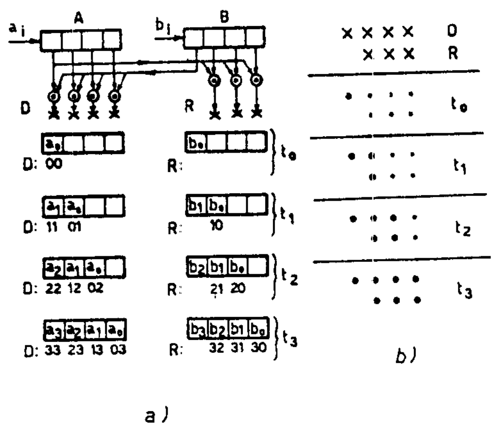
3



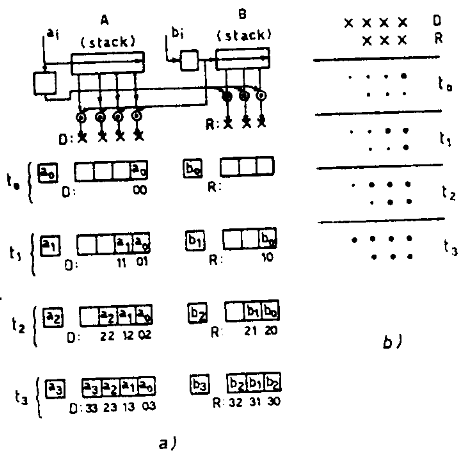
4


array generator output pins :
D0
R0
D1
R1
D2
R2
D3
R3
t0
t1
t2
t3

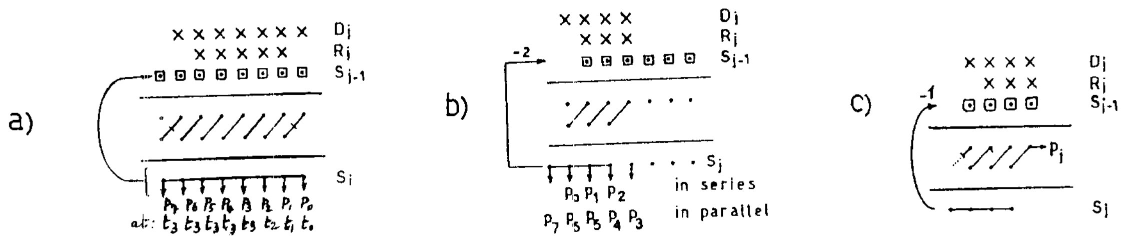
5


A
B
D
R
D: 00
D: 11 01
D: 22 12 02
D: 33 23 13 03
R: 10
R: 21 20
R: 32 31 30
a)
b)

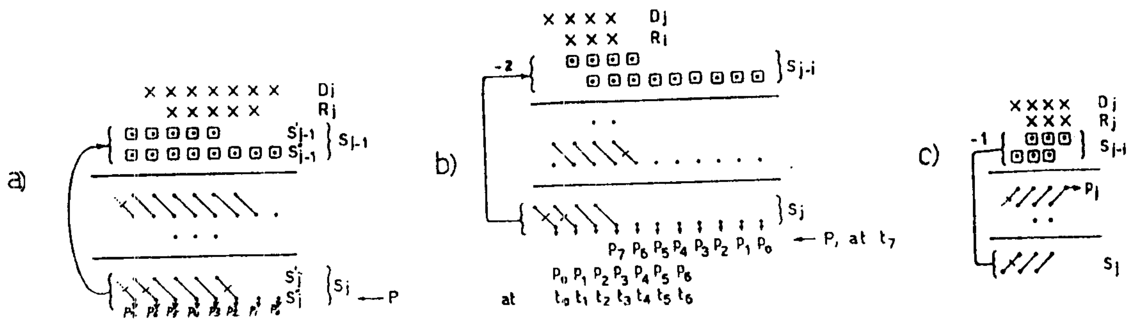
6


A (stack)
B (stack)
D
R
a)
b)

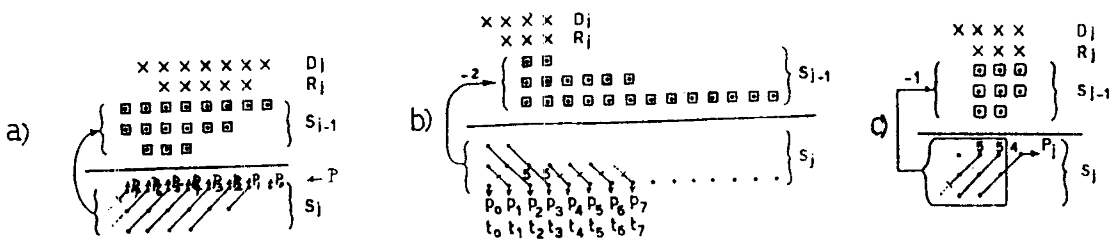
7


a)
b)
C)
Dj
Rj
Sj-1
Sj
in series
in parallel
Pj

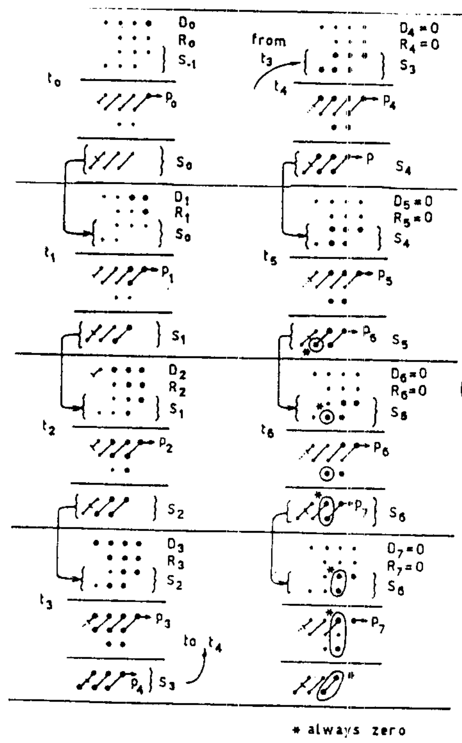
8


a)
b)
C)
Dj
Rj
Sj-1
Sj
P, at t7
at
Pj

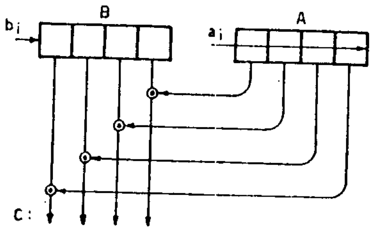
9


a)
b)
C)
Dj
Rj
Sj-1
Sj
P
Pj

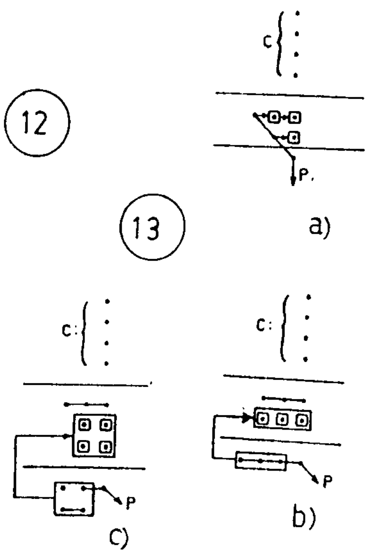
10



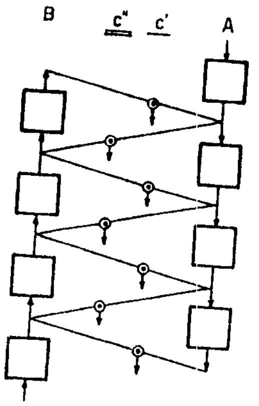
11



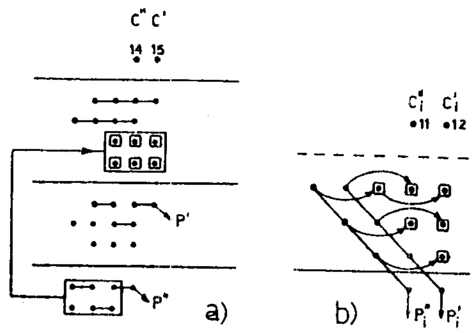
12



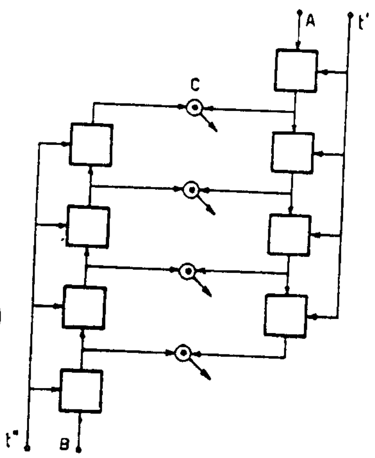
13



14



15



16

Fig. 1-The operation of the fastest serial input multiplier
Fig. 2-To show how the product array elements become available as operands bits are applied
Fig. 3-The graphical representation of various types of parallel counters
Fig. 4-A first scheme for the generation of the array elements
Fig. 5-To show how the new array elements generated at each step appear at the outputs of fig. 4
Fig. 6-A second scheme for an array generator: the outputs weights must be multiplied by four at each step

FIGURES CAPTIONS

Fig. 7-A third scheme for an array generator: the output weights must be multiplied by two at each step
Fig. 8-Schemes of summators with S_j represented by a single binary number
Fig. 9-Schemes of summators with S_j represented by two binary numbers
Fig. 10-Schemes of summators with S_j represented by three binary numbers
Fig. 11-To show the operation of fig. 9b summator
Fig. 12-A first scheme for the generation of the array by columns
Fig. 13-Schemes of summators for fig. 12 array generator
Fig. 14-A second scheme for the generation of the array by columns
Fig. 15-Schemes of summators for fig. 13 array generator
Fig. 16-A third scheme for the generation of the array by columns