SOME SCHEMES FOR PARALLEL MULTIPLIERS (*)

L. DADDA (**)

The possibility of combinational switching networks obtaining the product of two binary numbers is investigated. Some schemes are proposed, all based on the use of parallel counters, i.e. combinational switching networks which produce at the outputs in codified binary form the number of « ones » present at the inputs.

The proposed schemes obtain the product in two steps: the first obtains two numbers whose sum equals the product, without carry propagation; the second step obtains the product in a carry propagating adder.

The practical implementation of parallel counters is also considered.

1. – Introduction.

The realization of a parallel multiplier for digital computers has been considered in a recent paper by C. S. Wallace [1] who proposed a tree of pseudo-adders (i.e., adders without carry propagation) producing two numbers, whose sum equals the product. This sum can be obtained by applying the two numbers to a carry-propagating adder.

The purpose of this note is to present some schemes for parallel multipliers, based on a different principle and having some advantages over the one by Wallace. Also some of the proposed schemes will obtain two numbers whose sum equals the product.

2. – The multiplier scheme, based on parallel counters.

The new schemes are based on the use of logical blocks that we will call « parallel (n, m) counters »: these are combinational networks with m outputs and n ($\leq 2^m$) inputs. The m outputs, considered as a binary number, codify the number of « ones » present at the inputs. In a subsequent paragraph some implementations of such parallel counters will be illustrated.

Consider now the process of multiplication of two binary numbers, each composed of n bit, as been based on obtaining the sum of ν summands.

These summands are obtained, in the simplest schemes, by shifting left the multiplicand by 1, 2, 3, $(n-1)$ places, and multiplying it by the corresponding bits of the multiplier. In this case, $\nu = n$.

As it is well known, the number of summands can be made less than n by using some simple multiples of the multiplicand, on the basis of two or more multiplier digits [1].

The reduction of the number of summands will not be considered here. The case ($\nu = n$) will be therefore assumed, as the scheme proposed will work also for a reduced number of summands.

Consider now the case of two positive factors. To obtain the product, first represent the summands by the usual matrix as indicated in upper portion fig. 1; (in the figure, $n = 12$). In the same figure, the significant bits are

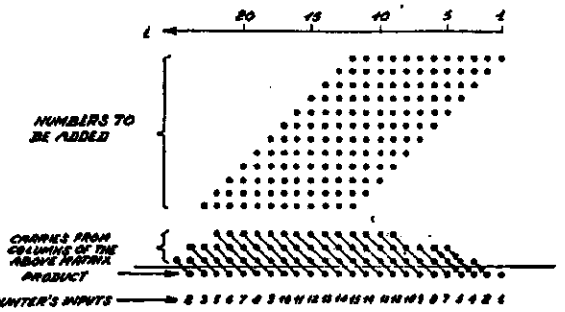

Fig. 1. — Multiplication (12 × 12 bit) through addition, in a single stage, using a parallel counter for each column. Carries are propagated through the counters.

represented by dots: moreover, bits at the left and at the right of the significant ones are supposed to be zeros.

The process of multiplication is as follows. The single bit in column $i = 1$ represents the least significant bit of the product, so it does not require any transformation. The two bits in column $i = 2$ are applied to a (2,2) counter: the least significant of the two outputs represents the second least significant bit of the product, and is recorded in fig. 1 on the last line, $i = 2$; the most significant output represents a carry, and is therefore recorded in column $i = 3$.

In column 3 we have 4 bit: three of them belong to the original matrix, the fourth is the carry just mentioned. This four bits will be applied to a (4.3) counter, whose three outputs will be recorded: in column 3 (the least significant, representing the third least significant bit of the product) in column 4 and in column 5 (the most significant), respectively.

The following columns are treated in a similar way, using suitable counters.

The inputs of each counter are in part the bits of the corresponding column of the summands matrix, in part carry bits produced by the counters of the preceeding columns. In fig. 1, carries produced by a given counter are connected by a diagonal segment.

The set of the least significant bits produced by all counters represents the result.

The above scheme is the most elementary one that can be devised. However, it suffers of a serious disadvantage, namely that of carry propagation delay through the counters.

(*) This paper has been presented at the « Colloque sur l'Algèbre de Boole », Grenoble, 11-17 january 1965.

(**) L. Dadda, Istituto di Elettrotecnica ed Elettronica del Politecnico di Milano.

(1) C. S. Wallace: *A suggestion for a fast multiplier*. - « IEEE Trans. on Electronic Computers », vol. EC-13, pp. 14÷17, febbruary 1964.

In fact, counters with a large number of inputs are in general implemented by complicated circuits, having therefore a substantial inherent delay.

If one wishes to minimize the effects of such delay, a different scheme should be adopted. This scheme is based on the following remark. Consider the problem of obtaining the product as divided in two steps. In the first step, obtain from the original set of addends a set of two numbers, whose sum equals the product. The second step obtains the product in a carry-propagating adder.

Carry propagation cannot be avoided: it is simply confined to this second step, where it can be accomplished by special, fast circuits.

Let us now describe the above process with reference to fig. 2, that represents a 12 × 12 product.

The first step of the process consists of cascaded stages (in the example, 3 stages): the first stage trans-

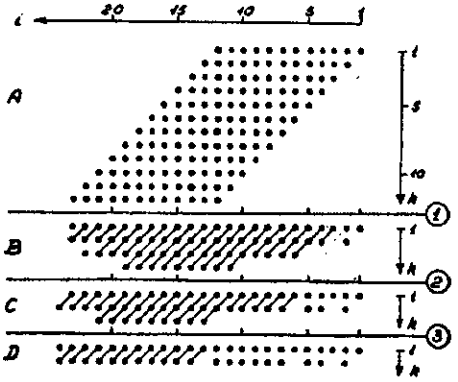

Fig. 2. — A multiplier scheme, obtaining two numbers, whose sum equals the product of 12 × 12 bit, through three stages, using a parallel counter for each stage and for each column. Carries are not propagated.

forms the matrix A into the matrix B; the second stage matrix B into C, and so on, until a matrix, composed of two rows only, is obtained: these two rows represent the numbers that, summed in a carry propagating adder, produce the result.

In the first stage, we begin by transcribing the single significant bit in column $i = 1$ (in the right upper corner) from matrix A into matrix B. The same is done for column $i = 2$, composed of two significant bits only.

The three bits of column $i = 3$, are applied to a (3,2) counter: the least significant, of the two output bits, is recorded in B, $i = 3$; the most significant bit is recorded in B, $i = 4$, in the second row. Which row is chosen for this second bit is inessential, provided it is recorded in column $i = 4$; the rule followed in Fig. 2 is convenient because it produces a compact matrix B.

The $i = 4$ column has 4 bit, and correspondingly, the parallel counter must have $m = 3$ outputs: the least significant bit will be recorded in B, $(i, k) = (3,1)$; the next bit in B, $(i, k) = (4,2)$; the third, most significant bit in B, $(i, k) = (5,3)$.

Note that in fig. 2 matrix B, the diagonal segment, joining together the above three bits, signifies that these bits are the outputs of the parallel counter fed by the column just above the least significant bit ($k = 1$). The rule is followed in all similar cases.

The process described above for column $i = 3$ and $i = 4$ is repeated for the subsequent columns, using each time an appropriate parallel counter.

The matrix B, thus obtained from A, has four rows: it is then transformed into matrix C by the same process.

Matrix C has 3 rows, so it is transformed into matrix D, a two-row matrix: the two rows represent the result of the process.

The above process can be justified as follows. With reference to the original matrix A, it can be said that the product is equivalent to the sum of all the bits $b_{i,k}$ in the same matrix, each multiplied by a weight, 2^{i-1}:

$$P = \sum_{1}^{n}{}_{k} \sum_{1}^{2n-1}{}_{i} b_{i,k}\, 2^{i-1}$$

The contribution to P of each column i, is:

$$P_i = \sum_{1}^{n}{}_{k} b_{i,k}\, 2^{i-1} = 2^{i-1} \sum_{1}^{n}{}_{k} b_{i,k}$$

This same contribution is represented in matrix B by the output of the counter, $\sum_{1}^{k}{}_{k} b_{i,k}$, whose least significant bit is in column i, and therefore has weight 2^{i-1}.

The matrix B, therefore, is equivalent to matrix A as far as the evaluation of the product is concerned.

Matrices C and D are also equivalent to A, having been obtained from B and C respectively by the same transformation.

The procedure described can be applied to factors with arbitrary value of n, on the assumption that parallel counters having a suitable number of inputs are available.

The number of stages required is easily determined, and appears as in table I.

TABLE I. – *Number of stages required for a parallel multiplier (vs. number of bits the multiplier) using parallel counters without limitation on the number of inputs.*

Number of bits in the multiplier	Number of stages
3	1
$3 < n \leq 7$	2
$7 < n \leq 15$	3
$15 < n \leq 32$	4
$32 < n \leq 64$	5

3. – The use of counters with a limited number of inputs: (2,2) and (3,2) counters.

Counters with a large number of inputs are difficult to realize. It is therefore important to see how the method could be applied using counters with a limited number of inputs, n_1.

It is apparent that the method can be modified as follows. If the number of significant bits in a column of the matrix to be transformed (initially matrix A) is greater than the number n_1 of inputs to the available counters, divide the bits in groups, each having at most n_1 bits. Each group can then be applied to the counter's inputs, each counter providing thus for each group the number of ones coded in binary.

Such counts are placed in the B matrix, all with the

least significant bit in column i, and using as many rows as necessary.

The same holds for the subsequent transformations, until a 2-row matrix is obtained.

The most important remark on the above procedure is that the number of stages necessary will in general be greater than that required with unrestricted counter inputs.

One might, in this connection, look for the minimum number of inputs, that can still be used for the implementation of the method.

This minimum number is of course *two*; (2,2) counters can be used as shown in the examples in Fig. 3, and Fig. 4, representing the cases $n = 3$ and $n = 4$ respectively.

Fig. 3. — A multiplier structure, obtaining two numbers, whose sum equals the product of 3 × 3 bit, through the use of 2 input/2 output parallel counters.

The example $n = 3$ works as follows. Columns $i = 1$ and $i = 2$ are reproduced in B. Column $i = 3$ is composed of 3 bit: therefore 2 of them are applied to a (2,2) counter, whose outputs are reproduced in B, columns $i = 3$ and $i = 4$ respectively (in the figure, they are connected together by a segment); the 3rd bit is simply reproduced in B, $i = 3$.

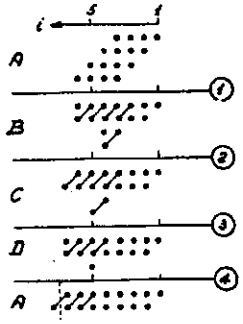

Fig. 4. — Same as fig. 3, for the product of 4 × 4 bit.

Columns 4 has 2 bit, as column 2: nevertheless it is transformed by a (2,2) counter, to produce the bits in columns 4 and 5 of B, because in B, column $i = 4$ there is already the carry from column 3. The single bit in column 5 is simply reproduced.

The case $n = 4$ is completely analogous. Considering cases with larger n, it can be found that the number of necessary stages, when using (2,2) counters, is 2^{n-2}. This means that, for pratical values of n (e.g., $n = 30$), the number of stages would become very large and consequently the total delay would become too great.

As we will see in the following, the number of stages is drastically reduced if counters with $n = 3$ or more are used. On the other hand, it must be noted that counters with n larger than 2 are not difficult to implement. This is certainly the case for $n = 3$, that is a full adder network.

Fig. 5 represents the multiplication process for the case $n = 12$ (like fig. 2), using (3,2) counters and, when necessary, also some (2,2) counters. It can be seen that the total number of stages required is 5 (instead of 3,

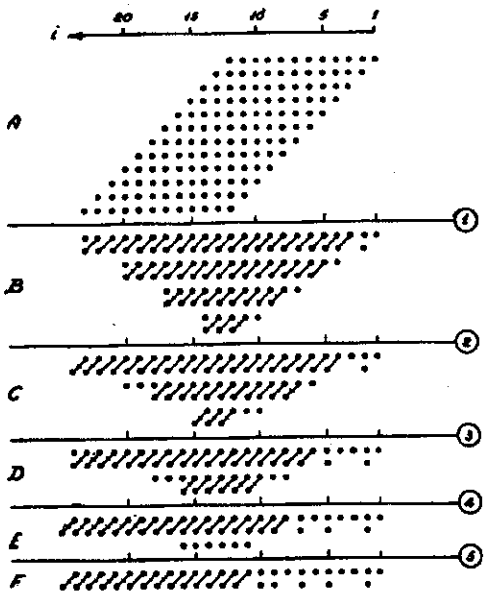

Fig. 5. — Same as fig. 2, through the use of 3 or 2-input/2 outputs parallel counters.

which represents the absolute minimum; see fig. 2). The process is carried out in fig. 5, with the following rules.

Columns 1 and 2, are simply reproduced in B (and in all following matrices).

Column 3, having 3 bit, is transformed in B into 2 bit: the least significant in column 3, the most significant in column 4 (second row). (They are connected by a diagonal segment, as used previously).

Column 4 has 4 bit: three of them are transformed by a (3,2) counter, the fourth is simply reproduced in B, (4,3).

Column 5 has 5 bit: three of them are transformed by a (3,2) counter, the last two by a (2,2) counter: the latter's outputs (in B, (5,3) and B, (6,4), respectively) are joined by a diagonal segment crossed by a bar to signify that they are outputs of a (2,2) counter.

Similar rules are applied to all the following columns, through the last column (23rd) which has a single bit.

The matrix B obtained by the application of the process, can be shown to be equivalent to A, as far as the evaluation of the product is concerned.

The same process can be used to obtain C from B, D from C, E from D and F from E. F has two rows, whose sum represents the product.

Some remarks can be made about the described process, which was based on (3,2) counters.

a) the number s of stages can be determined as follows.

It can be seen that the last 2-row matrix can be derived by a 3-row matrix. This is true regardless of the type of counters used.

A 3-row matrix can be derived from a 4-row matrix: in fact, three of the bits of each column can be reduced to two, by a (3,2) counter; the fourth bit is simply reproduced. This is obtained in the $(s - 1)\,th$ stage.

A 4-row matrix can be derived from a 6-row matrix, as can be easily verified. This is the $(s-2)$ t h stage.

A 6-row matrix can be derived from a 9-row matrix: $(s-3)$ t h stage.

A 9-row matrix can be derived from a 13-row matrix: $(2+2+2+2+1 \rightarrow 3+3+3+3+1)$. This is the $(s-4)$ t h stage.

In the example of fig. 4, $n = 12$, so that the number of stages is given by: $s - 4 = 1$; $s = 5$, as obtained.

Proceeding with the same rules, table II, valid for (3,2) counters, can be drawn up.

TABLE II. – *Number of stages required for a parallel multiplier (vs. number of bits of the multiplier) using (3,2) counters only.*

Number of bits in the multiplier	Number of stages
3	1
4	2
$4 < n \leq 6$	3
$6 < n \leq 9$	4
$9 < n \leq 13$	5
$13 < n \leq 19$	6
$19 < n \leq 28$	7
$28 < n \leq 42$	8
$42 < n \leq 63$	9

b) The scheme of fig. 5 is not the only one possible with (3,2) counters.

Fig. 6 represents a process, sligtly different from that in fig. 5 and leading to a considerable saving in components. In fact, the scheme in fig. 5 requires a total of

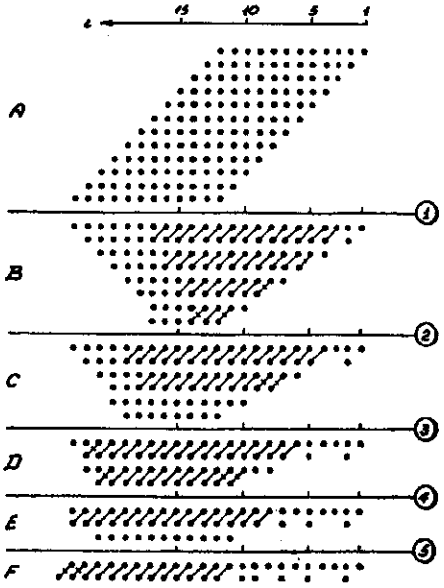

Fig. 6. — Same as fig. 5, with a different scheme using less counters.

136 counters (101 are (3,2) counters, 35 are (2,2)); the scheme of fig. 6 requires 116 counters (104 are (3,2), 12 are (2,2)).

The rules applied in fig. 6 are the following.

In the first stage, columns 1 to 13 are treated in the same way as in fig. 5.

Columns 14, 15, 16 and 17 (having 10, 9, 8, 7 bit respectively in matrix A, are only partially reduced, so that in matrix B they are composed of 8 bit (taking into account the carries from the preceeding columns).

All remaining columns are reproduced in matrix B without any reduction.

The reason of doing so is essentially the following. It is not convenient to try reduction of the number of bits in a given column, when it is preceeded by columns having a larger number of bits, because carries from the latters tend to increase the number of bits.

It should be remarked that in passing from matrix B to matrix C the last two rows of matrix B are reproduced in C without any transformation.

c) A sligtly different criterion is applied in fig. 7,

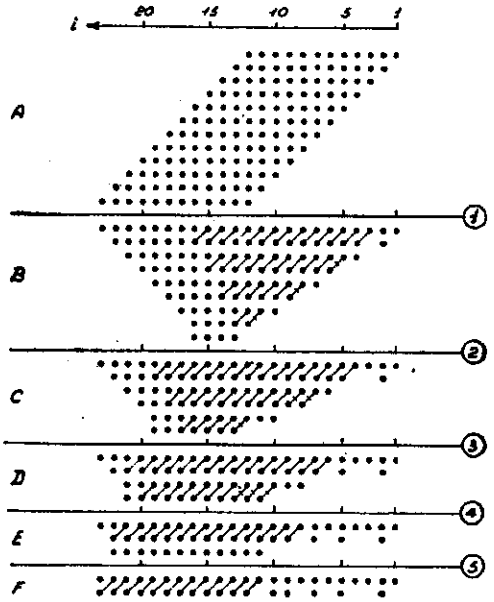

Fig. 7. — Same as fig. 6, with a different scheme, using less counters.

where a further reduction in the number of counters is achieved (113 instead of 116).

The scheme in fig. 7 is the same as in fig. 6 for columns 1 to 12.

Column 13 and the followings, are only partially reduced so that in matrix B they have no more than 9 bit (while in Fig. 6, 8 is the maximum number of rows in matrix B.

Although nothing can be said in general about the effect of such a rule, it can however be noted that both 9-bit columns and 8-bit columns are reduced successively to 6, 4, 3, and 2 bit in the succeeding matrices. The reduction of such columns from matrix A to matrix B requires less counters for 9 column than for 8. Moreover, although in the succeeding stages more counters are required for a 9-row than for an 8-row B matrix, there is a net saving in the total number of counters.

d) Another reduction scheme can finally be described, as the example in fig. 8 shows. It requires the least number of component of all schemes considered: 110 counters (96 of (3,2) type; 14 of (2,2) type).

The rules applied in Fig. 8 can be described as follows. First, notice that in the original matrix A the middle column (12th) has 12 bit, and that proceeding from that

columns to the right and to the left, columns have a decreasing number of bits.

In passing from matrix A to matrix B, columns are only partially reduced, so that no more than 9 rows are obtained. For example, column 10 (10 bit) is trans-

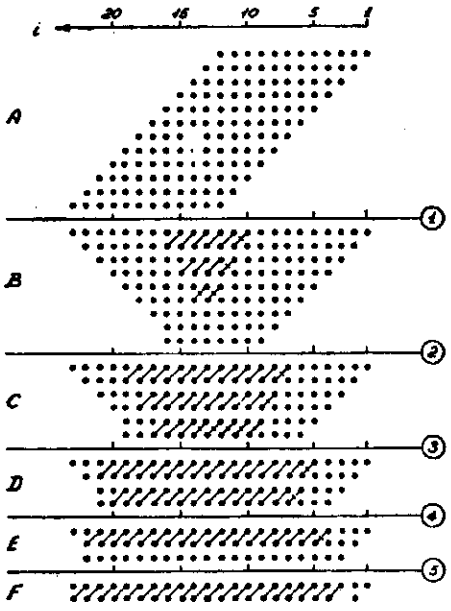

Fig. 8. — Same as fig. 7, with a different scheme, using less counters.

formed in a 9-bit column in B, by reproducing 8 bit without transformation and transforming only 2 bit, by a (2,2) counter.

Consequently, only some columns in the central portion of matrix A are actually transformed.

In passing from matrix B to C, columns having no more than 6 bit are obtained. In succeeding transformations, columns with no more than 4, 3 and 2 bit respectively are obtained.

The above rules can be generalized for $n \times n$ bits multiplications as follows.

Consider first the following series:

$$2;\ 3;\ 6;\ 9;\ 13;\ 19;\ 28;\ 42;\ 63;\ \ldots$$

(where each term is obtained from the preceeding, by multiplying it by 3/2 and taking the integral part). The terms of this series correspond to the number of those matrices obtained from the final, 2-row matrix, and applying the reverse of the transformation described in the preceeding examples.

Given then the original A matrix for a $n \times n$ bits multiplication, obtain through the first transformation a matrix B having a number of rows coinciding with the nearest term of the above serie which is less than n.

All the following matrices will have a number of rows coincident with the terms of the series (in decreasing order of magnitude).

From the above examples it appears that the best rule is the last one described.

Although no proof is given here of the optimality of such rule, nevertheless all examples worked out for different values of n are in accordance with the results obtained for $n = 12$.

e) It is interesting to compare the described schemes with the Wallace scheme. This can be considered as a parallel multiplier composed of (3,2) counters.

The Wallace multiplier is based on a tree of pseudo-adders, as shown in the block-diagram of fig. 9 (Wallace notation). Each pseudo-adder is effectively composed of a set of (3,2) counters, as appears in fig. 10, where the notation used in the preceeding figures of this paper is used.

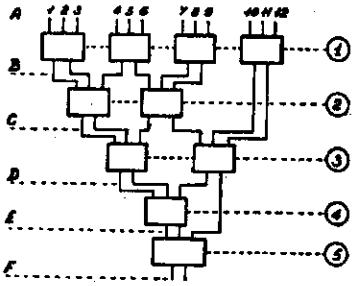

Fig. 9. — Block diagram for a parallel (12 × 12) multiplier structure, according to Wallace.

Fig. 10, concerning the case $n = 12$, requires a total of 136 counters (102 are (3,2), 34 are (2,2)), that is the same number of counters required in the fig. 5 scheme. This coincidence is not valid in general. It can be shown that, for $n > 12$ the Wallace scheme requires less counters than the fig. 5 scheme.

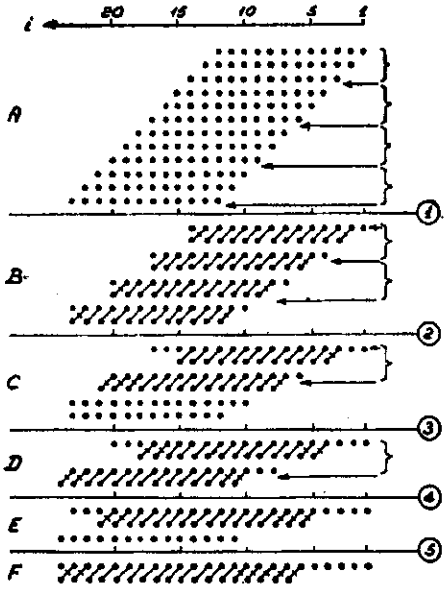

Fig. 10. — The Wallace scheme for a (12 × 12) multiplier.

However, the schemes based on the rules illustrated in the preceeding paragraph *d* (see fig. 8) requires less counters than Wallace scheme.

For instance, for the case $n = 24$ one obtain:

— Wallace scheme: 575 counters
— fig. 8 scheme: 506 counters
— fig. 5 scheme: 606 counters.

f) The following remarks can be made about the described schemes. In all the examples (fig. 5, 6, 7, 8)

it can be seen that the bits in the least significant portion of the result are produced through a number of stages smaller than the total.

Examining, for instance, fig. 5, it can be verified that:

— the least significant bit ($i = 1$) and the two bit in column $i = 2$ are produced without any stage.

— the bit in column $i = 3$ is produced through the first stage;

— the bit $i = 4$ is produced through the second stage; etc. etc.

This is very important if the speed of the circuit is considered, as each stage introduces a certain delay.

In fact, the above remark means that the least significant portion of the two final numbers is produced with a delay that increases progressively from the least significant bit (no delay) on.

Because the two final numbers must be added together, to obtain the product, in a carry-propagating adder, this means in turn that the carry propagation delay in the least significant half of the adder is overlapped by the progressively increasing delay through the multiplier structure. Whether the carry delay in the adder is greater than the multiplier delay or not, depends of course from the type of circuit used and also from the operands.

It can be seen also in the examples given (see e.g. fig. 5) that the second final summand (second row in matrix F) has some zeros, regardless of the bits of the original operands. Such situation can be accounted for in the construction of the adder, using half adders in the stages corresponding to the zeros of the second summand, thus simplying the final adder.

4 – The use of higher order counters: (7,3) and (15,4) counters.

The (3,2) counter, i.e. the full adder, is the commonest form of implementation of the parallel counters concept, and the use of such counters in parallel multipliers has been discussed in the preceeding point.

Higher order counters, i.e. counters having a larger number of inputs and outputs, although not currently used, are nevertheless entirely feasible with today's technology, as will be shown in the next paragraph. This is certainly the case for (7,3) counters. Moreover, high order counters compare very favorably with (3,2) counters as far as the number of components is concerned. It is therefore interesting to investigate briefly on the problem of multiplier's implementation using such counters.

It can be shown that most of the considerations illustrated in the preceeding paragraph are valid for higher order counters. In particular, the scheme illustrated in paragraph *d*) proves to be the best, as far as the total number of counters is concerned.

The most important point to be illustrated is the number of stages required, as a function of the counters order.

Suppose that, beside (3,2) counters, (7,3) counters are available. Starting now from the final 2-row matrix, observe that it can be obtained from a set of 2-output counters, i.e. (2,2) or (3,2) counters. This means that the next matrix must have only 3 rows.

A 3-row matrix can be obtained from a set of 3-output counters, i.e. from counters having 7 inputs, at most. The next matrix must therefore have 7 rows at most.

A 7 row matrix can be decomposed in two 3-row matrix and a 1-row matrix. This means in turn that it can be derived, through (7,3) counters, from a matrix having $7 + 7 + 1 = 15$ rows.

Proceeeding with such rules, the following series can be obtained:

$$2;\ 3;\ 7;\ 15;\ 35;\ 79;\ \ldots$$

that can be used as the series of the preceeding paragraph (valid for (3,2) counters).

For example, if a multiplier is to be designed for numbers having 48 bit, it can be seen that 5 stages are required. The first stage will obtain a 35-row matrix, the second a 15-row matrix, etc.

If now we suppose that (15,4) counters are available, beside (3,2) and (7,3) counters, the following series can be obtained, using rules similar to those applied for the preceeding case:

$$2;\ 3;\ 7;\ 21;\ 61;\ 226;\ \ldots$$

It can be seen therefore, that for a 48 bits multiplice, 4 stages will be required.

As was announced previously, high order counters can afford an important saving in components. For example, if counters based on threshold devices are used (see next paragraph), the total numbers of transistors, required by the multiplier structure for 24 bit (excluding the carry propagating adder and the network generating matrix A) is as follows:

(3,2) counters:	Wallace scheme:	1150 transistors
	scheme *d*):	1012 transistors
(7,3) counters:	scheme *d*):	490 transistors

5. – Remarks on parallel counters.

The schemes discussed in the preceeding paragraphs are all based on the use of parallel counters. It is therefore worthwhile to discuss briefly on their practical implementation.

Before describing some counters, let us discuss on some characteristics that prove useful in the peculiar application considered.

Among the different type of full adders, the most suitable for the application in parallel multipliers, from the point of view of economy and speed, are those which require input variables of one form only (natural or complemented), so that output variables of the same form only must be generated. If such condition is satisfied, outputs of one stage can be used directly as inputs to the next stages, without the need of inverters, leading in general to a considerable saving in components and to a reduction of stage delay.

It must be noted that the above restriction can be partially released by allowing the use of counters producing outputs of only one form but different from the input's form (« inverting counters »). We will examine later some simple circuits, of this type.

The use of inverting counters does not modify substantially the described schemes. Consider the use of such counters in the scheme of fig. 5 and suppose that all the bits of the final matrix F are to be obtained in true form. Assume then the preceeding matrix E to be in complemented form, the preceeding matrix D in

true form, etc.: i.e., matrices are alternatively in true or complemented form. This situation can be obtained if, when transforming from one matrix to the next, one use inverting counters or, when bits were simply to be reproduced, inverters.

All inverters can nevertheless be avoided, if single bits in the original matrix A are produced in a suitable form. Consider for example, columns 1 and 2: according to the previous rule they should be in complemented form in matrix A: but they can be simply transfered from A to F if they are produced in A in true form. On the contrary, column 3 must be produced in A in complemented form, so that the single bit produced in matrix B, column 3) in true form, can be directly transfered to F, and so on.

It can thus be said that inverting counters can be used, provided that bits in the original matrix A are produced in a suitable form.

An interesting feature of all schemes of parallel multipliers is that each counter output is loaded by a single input of a counter of the next stage. This feature can be conveniently accounted for in the electrical design.

Some considerations will now be made on the implementation of parallel counters, with reference to some of the available logic circuitry and taking into account the above remarks.

a) « *And-or-not* » logic. A full-adder, satisfying the above requirement, can be based on the following equations:

$$R = A\,B + A\,C + B\,C$$
$$S = R\,(A + B + C) + A\,B\,C$$

where:

A, B, C, are the input variables

R, S are the outputs.

A full adder of this type has been reported by Wallace [1]. It is an inverting counter, requiring 18 diodes and 3 transistors, and having a total delay of 60 ns.

It does not seem that the realization of counters of higher order, and satisfying the above requirement, has been investigated. Nevertheless it can be said that the number of components will increase rapidly along with n.

Similar conclusions can be accepted for « nor » (or « nand ») logic, although cheaper circuits will be obtained.

b) *Threshold gates.* Counters using threshold gates can be of the non-inverting type or of the inverting type

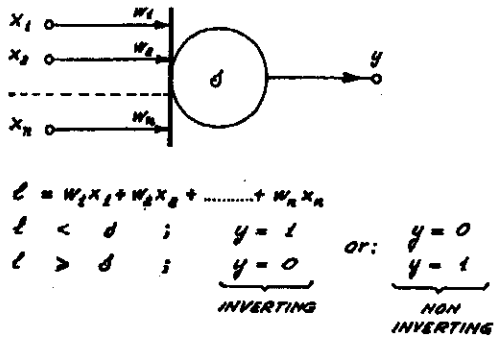

Fig. 11. — The logical definition of threshold gates.

according to the type of threshold gate used (see fig. 11). An interesting scheme for inverting counters, is reported in Fig. 12 (for a (3,2) counter), and Fig. 13 (for (7,3) counter.

A simple realization of an inverting (3.2) counter using resistor-transistor gates, is represented in fig. 12,*b*.

It is probably the simplest circuit that can be devised for a full-adder, as it uses only 2 transistors (one for each output) and some resistors. Using available components, a delay of less than 100 ns can be obtained.

An investigation has been undertaken in our laboratory in order to explore the possibilities of threshold counters for parallel multipliers.

c) *Current switching.* Current switching circuits offer a mean for implementing parallel counters. Current switching can be realized using transistors or criotrons. Transistor current switching circuits are the fastest logical circuits realizable with a given transistor type [2].

Using available transistors, it should be possible to

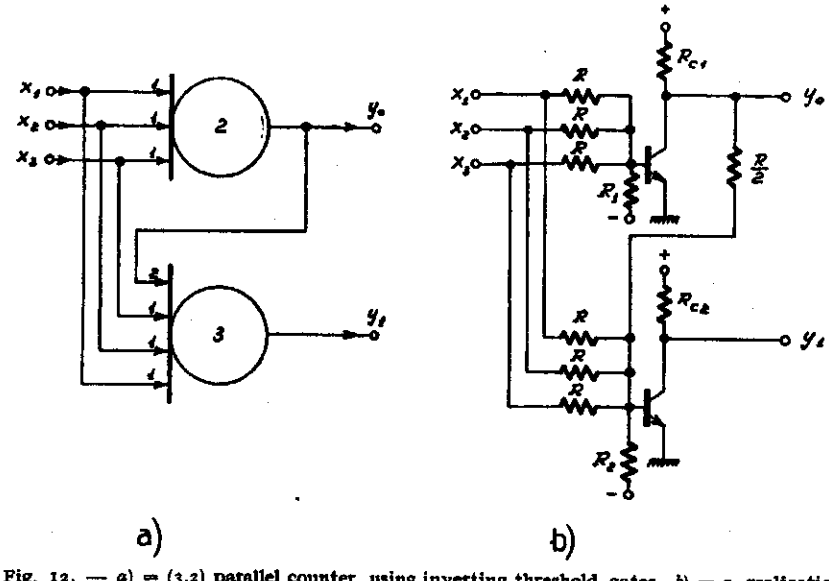

Fig. 12. — *a*) = (3,2) parallel counter, using inverting threshold gates. *b*) = a realization of an inverting (3,2) counter, using resistor-transistor threshold circuits.

realize (3,2) counters having a delay of 10÷20 ns. They are, however less economical than all other circuits.

It should be noted that special circuits (i.e. not suitable for general purpose logic) could probably be devised in order to obtain fast and cheap counters. A similar situation is afforded by parallel adders, that can be realized using expecially designed circuits (see: [2]).

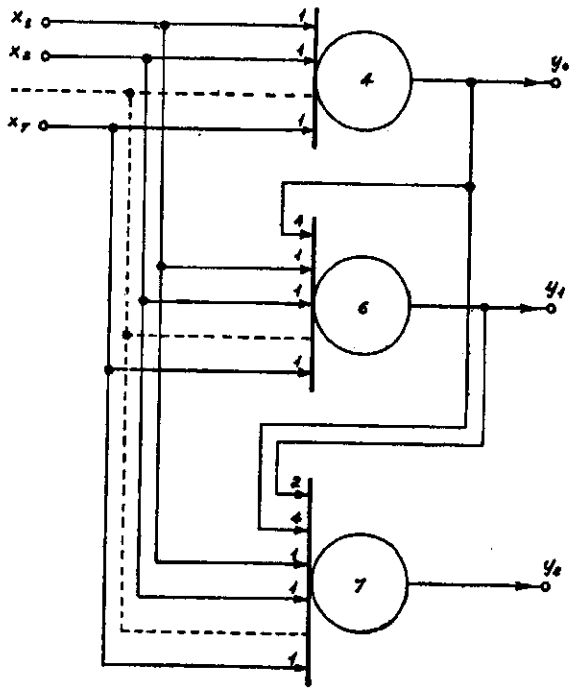

Fig. 13. — a (7,3) parallel counter, using inverting threshold gates.

6. – Conclusions.

Having established the possibility of a parallel digital multiplier, some considerations can now be made about the important aspects of speed and cost than can be encountered in a practical design.

It is worthwhile first to recall that if one assumes [1] that a third of all arithmetic operations in scientific computers are multiplications and that these at present take about four times as long as additions, the use of a fast multiplier allowing a multiplication in a memory cycle time, would approximately double the speed of computation.

There is therefore a chance that a parallel multiplier could become a convenient mean to improve the value of a computer, owing to the fact that it cost can be shown to be only a few percent of the total computer cost.

The following is an estimate about the type of multiplier circuits, based on actual memory cycle times.

Let us first note that the total multiplication time is composed of two parts: the first is the time elapsed from the application of the signals representing the two factors to the inputs of the multiplier, to the availability of the inputs to the carry-propagating adder; the second part is the delay proper of the adder, mainly consisting in the carry propagation delay.

In the design of a practical multiplier, one can assume as a goal to obtain a total delay equal or less that to the cycle time of the high-speed memory, so that the computer can work at its maximum speed, limited only by the memory speed. The choice of the type of circuits depends therefore from the memory cycle time.

If a core memory having a cycle of 4 μs or more is considered, threshold gates allow a very convenient solution. The fastest core memories have cycle time in the order of 1 μs. In this cases, probably threshold gates could again be used, provided (3,2) counters having delays of the order of 50 ns are designed and carry propagating adders with less than 100 ns delay are employed.

In cases where fastest memories are considered (for instance, magnetic film memories, or tunnel diodes memories) having cycle times of 200 ns or less, fastest counters should be designed, for instance of the current switching type.

Although the problem is beyond the scope of this paper, it must also be noted that the realization of parallel multipliers should also influence computer organization. It is well known, indeed, that some important features of fast computers depend on the fact that during operations, that last longer than one memory cycle (typically, during multiplication or division), memory can be made available for other operations (e.g. input-output).

It appears therefore necessary to review the computer structure, as far as it depends from the duration of multiplications.

The paper was first received 29th *Avril* 1965.

([2]) D. B. Jarvis, L. P. Morgan, J. A. Wearer: *Transistor current switching and routing techniques.* - « IRE Trans. on Electronic Computers », vol. EC-9, n. 3 pag. 302, september 1960.