FAST MULTIPLIERS FOR TWO'S-COMPLEMENT NUMBERS IN SERIAL FORM

Luigi DADDA

Department of Electronics - Politecnico di Milano
P.zza L. da Vinci 32, I-20133 Milano - Italy

ABSTRACT

Schemes for designing multipliers of binary two's-complement numbers in serial form are considered with the condition of the least possible delay between inputs and output.

Such schemes are composed by two parts: the first, the array generator, produces the terms of the multiplier array; the second, the summer, is fed by the array generator and produces the product. Two classes of multipliers are illustrated: the first generating the multiplier array by diagonals and rows, the second by columns.

The array generators are composed by shift and/or stack registers and linear arrays of logic gates; the summer is shown to be conveniently built using parallel counters.

INTRODUCTION

Fast digital multipliers are extensively used, particularly in special purpose computers, for applications requiring frequent multiplications to be performed very quickly, as in signal processing.

Many different schemes have been proposed for implementing fast multipliers. The various schemes can be classified according to the form in which factors are available. If both factors are in parallel form, fully parallel multipliers can be used [1]; if they are in serial form (with the least significant bit appearing as first) they require serial multipliers [1]; if one of the factors is in parallel, the other being in serial form, the corresponding multiplier is called serial-parallel (or, by Swartzlander [2], quasi-serial).

In a previous article [11] the author has treated the case of serial multipliers for sign-and-absolute value numbers, showing that they can be designed according to various criteria. In this note, it will be shown how such schemes can be modified in order to handle factors represented as two's-complement numbers.

This work was partially supported by European Economic Community in the frame of Project CVT

In the following section the arithmetic algorithms for treating two's-complement multiplication will be recalled, and in the third paragraph it will be shown how such algorithms can be implemented in the schemes described in [11].

ARITHMETIC ALGORITHMS

An n bits integer X in two's-complement will be represented as

$$X = \sigma_x x_{n-2} \cdots x_1 x_0 = -\sigma_x \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

where the most significant bit is denoted by σ_x, as it gives information about the sign of X:

$$\sigma_x = 0 \quad \text{for } X \geq 0$$
$$\sigma_x = 1 \quad \text{for } X < 0$$

the range of X is:

$$-2^{n-1} \leq X \leq 2^{n-1} - 1$$

The following properties of two's-complement numbers will be useful in the following:

- to change sign, complement each bit and add '1';
- X can be extended to arbitrary length by repeating σ_x as many times as needed ("sign extension"):

  e.g. $\sigma_x x_2 x_1 x_0 = \sigma_x \sigma_x x_2 x_1 x_0 = \sigma_x \sigma_x \sigma_x x_2 x_1 x_0 = \cdots$

  In such case the leftmost σ_x has a negative weight (-2^{n-2+K}) if K σ_x are used; the remaining σ_x (if any) have positive weight corresponding to their position;
- in adding m, n-bit numbers, they can be added by columns regardless of their respective sign, starting from the least significant bits column and proceeding to the left, the signs column included. The range is not exceeded (i.e. no overflow occurs) if, being ν the numbers of the negative addends and C_{n-1} the total carries from the (n-2)th column to the (n-1)th column (the σ_x column):

  $C_{n-1} = \nu$: the result is positive

  $C_{n-1} = \nu - 1$: the result is negative

Note that the sum can be accommodated by extending its length with a suitable number of bits.

CH2146-9/85/0000/0057$01.00 © 1985 IEEE

Note also that if the addends have unequal length, they must be brought to the same length (the length needed to accommodate the sum) by sign extension (see the preceding rule).

The product

$$P = X.Y$$

of two, two's-complement numbers can be computed by: first, obtaining the usual "multiplier array", see fig. 1, second, adding the n numbers (rows) comprising such an array.

It must be noted that the length of a product P of two n bits natural numbers is 2n. (If X and Y are in sign and magnitude form, a further bit to represent the sign of P, σ_p is to be added: if the convention $\sigma = 0$ for positive numbers, $\sigma = 1$ for negative is adopted, then $\sigma_p = \sigma_x \oplus \sigma_y$).

For two's-complement n-bits numbers, the product length necessary to accomodate the total range of P is 2n bits, including the sign σ_p. But if the negative limit -2^{n-1} of the X and Y range is restricted to $-2^{n-1}+1$, 2n-1 bits will be sufficient for P. The following schemes will show both alternatives.

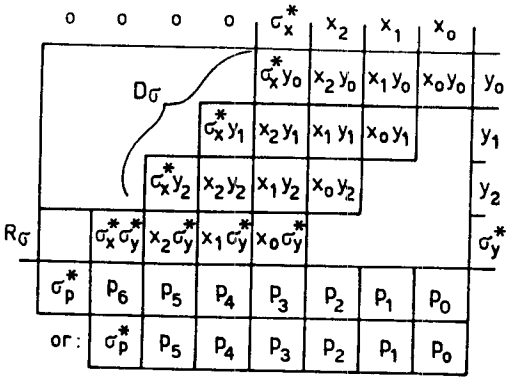

Fig.1 : Multiplier array for two's-complement numbers of four bits each.
σ_x^* and σ_y^* have negative weight.

In fig. 1 each term in the array is the arithmetic product of two bits belonging to X and Y. While in terms $x_i.y_j$ the factors x_i and y_j being affected with positive weights, the arithmetic product coincides with the logical product, in terms $\sigma_x.y_j$, $x_i\sigma_y$ and $\sigma_x.\sigma_y$ it must be taken into account the negative weights of σ_x and σ_y. For this reason, in fig.1 σ_x and σ_y have been written with an asterisk for denoting their negative (or zero) weight. Note that $\sigma_x.\sigma_y = \sigma_x^*.\sigma_y^* = \{0, 1\}$.

The addition of the array terms, in order to obtain the product P would, then, to consider the negative terms comprising the 'row' R_σ and the 'diagonal' D_σ. It seems preferable to transform the array in such a way that it contains all positive terms (being of course equivalent to the original array). This can be done in several ways: among them, the following appear as the most convenient for the purpose of designing a multiplier for numbers in serial form :

- The first way is to extend X and Y, by their σ, and obtaining then the multipliers array for the extended X and Y, see fig. 2.

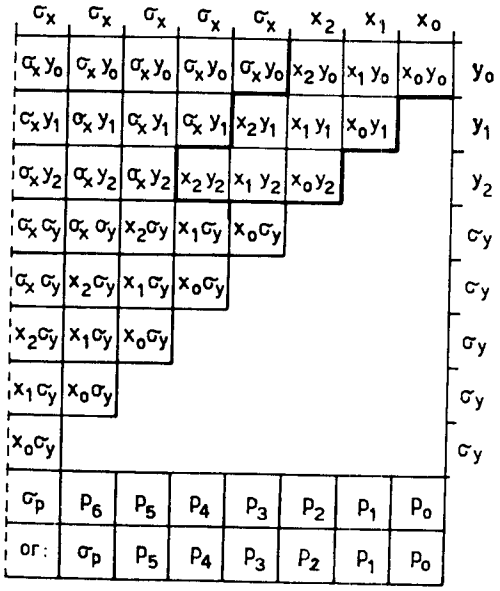

Fig.2: A multiplier array for two's-complement numbers of four bits, composed of non-negative terms only, obtained by extending the sign bits σ_x, σ_y.

The array can then be limited to the leftmost 2n-th (or 2n-1)th) column, the columns to the left being irrelevant for the computation of P (if more columns were used, more σ_p would be obtained);

- A second way of transforming the original array is to replace R_σ and D_σ (containing zeros and negative terms) with equivalent numbers comprising only zeros and positive terms.

  This can easily be obtained, as the following example shows:

  $$0\ -1\ -1\ 0\ -1 = (-1)\times(0\ 1\ 1\ 0\ 1) =$$
  $$= 1\ 0\ 0\ 1\ 0 + 1 = 1\ 0\ 0\ 1\ 1$$

  In words: a binary integer composed of 0 and -1 bits is equivalent to the negative of a number obtained from the given number by changing -1 into 1.

By applying the above rules to D_σ one obtains:

$$D_\sigma = \ldots 0 \quad 0 \quad \sigma_x^* y_2 \quad \sigma_x^* y_1 \quad \sigma_x^* y_0 =$$
$$= \ldots 1 \quad 1 \quad \overline{\sigma_x y_2} \quad \overline{\sigma_x y_1} \quad \overline{\sigma_x y_0} + 1$$

R_σ is a binary number in which the most significant bit: $\sigma_x.\sigma_y = \{0,1\}$, and the remaining bits are: $x_i\sigma_y = \{0,-1\}$. The latters, which compose the part R'_σ of R_σ with negative weights, can be transformed as done for D_σ :

$$R'_\sigma = \ldots 0 \quad 0 \quad x_2\sigma_y^* \quad x_1\sigma_y^* \quad x_0\sigma_y^* =$$
$$= \ldots 1 \quad 1 \quad \overline{x_2\sigma_y} \quad \overline{x_1\sigma_y} \quad \overline{x_2\sigma_y} + 1$$

The sum of the leading 1's of D_σ and of R'_σ is: ...1 0, so that the new array, composed by non-negative terms only, appears as in fig. 3. Note that the two "1" to be added to the least significant bits of D_σ and R'_σ are replaced by a single "1", one column to the left.

The above method is similar to the one proposed by Baugh and Wooley [4] for parallel multipliers.

weights:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
					$\overline{\sigma_x y_0}$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$
				$\overline{\sigma_x y_1}$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$	
			$\overline{\sigma_x y_2}$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$		
		$\sigma_x \sigma_y$	$\overline{x_2 \sigma_y}$	$\overline{x_1 \sigma_y}$	$\overline{x_0 \sigma_y}$			
	1			1				

Fig.3 : An equivalent array composed by non-negative terms only.

A third form of equivalent array is represented in fig.4, where the first three rows of fig. 1 have been transformed by sign extension (thus taking account of D_σ).

As far as R_σ is concerned, note that it can be considered as a binary two's-complement number, where the signs of the weights have been reversed. The sign bit, $\sigma_x \sigma_y$, can thus be extended indefinitely. Transforming then the five least significant bits (the only ones affecting the product), the result is as in fig. 4. Note that the corrective "1" for R_σ can be represented by the group of "1" enclosed in the dotted line in the figure.

Several more variations of the arithmetic algorithms illustrated can be devised, see [5]. The ones shown here have been found to be the most suitable for the implementation of serial, two's-complement multipliers.

MULTIPLIERS SCHEMES

It has been shown in [11] that serial multipliers can be composed by cascading two networks: the first, (the array generator) fed by the factors bits, generates the array elements, which are input to the second network (the summer), generating the product by summing the array elements received.

Two classes of multipliers can be devised, characterized by the method used for generating the array elements.

The first class generates, for each new couple of factors bits, the corresponding (partial) "row" and (partial) "diagonal": they contain all the elements that can be computed using the new factors bits and the factors bits which have been previously input: see fig. 1.

weights:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	$\sigma_x y_0$	$\sigma_x y_0$	$\sigma_x y_0$	$\sigma_x y_0$	$\sigma_x y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$
	$\sigma_x y_1$	$\sigma_x y_1$	$\sigma_x y_1$	$\sigma_x y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$	
	$\sigma_x y_2$	$\sigma_x y_2$	$\sigma_x y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$		1
	$\overline{\sigma_x \sigma_y}$	$\overline{\sigma_x \sigma_y}$	$\overline{x_2 \sigma_y}$	$\overline{x_1 \sigma_y}$	$\overline{x_0 \sigma_y}$	1	1	1
					1			

Fig.4 : An equivalent multiplier array composed by non-negative terms only.

The second class of multipliers is based on generating the array "by columns", (starting from the rightmost, 1-element column $x_0 y_0$).

Various schemes can then be proposed for each class [11]: only some of them will be considered here, for sake of brevity and because the criteria explained in the following can be easily extended to the remaining schemes.

Let us first consider the case of multipliers based on rows and diagonals generators.

One of such generators (implementing the array of fig. 3) is represented in fig. 5: it consists of two shift registers feeding a first linear array of AND gates. A second array of gates acts in such a way that for $t_1 \neq t_\sigma$ (t_σ is the bit time for the factors sign bits; in the example $t_\sigma = t_3$) the outputs of the first array are reproduced at the R and D outputs, the outputs of the array generator. At time t_σ, the outputs of the AND-gates array appear complemented at the outputs R and D. (The AND array and the inverting array have been considered as distinct for reason of clarity: of course, they can be implemented as a single, equivalent array).

In fig. 5b the outputs D and R having the same weight are represented on the same column, and the dots represent the significant array terms generated at various clock times.

It can be seen, by comparing fig. 5 with fig. 3 that at each step, the weights of the R and D outputs must be multiplied by four in order to have the correct weights (see[11]). This will be accounted for in the following summer circuit, whose logical scheme is represented in fig. 6a (with symbols and conventions as used in [11]) and whose purpose is to sum-up the R and D bits generated by the array generator, and to produce the product bits. The first two rows represent the D and R outputs of the array generator; the third row contains two inputs fed by a logical 'one' at time t_3: it represents the two "1" in the scheme of fig. 3.

The following three rows represent memory element used to store the carries produced at the various steps.

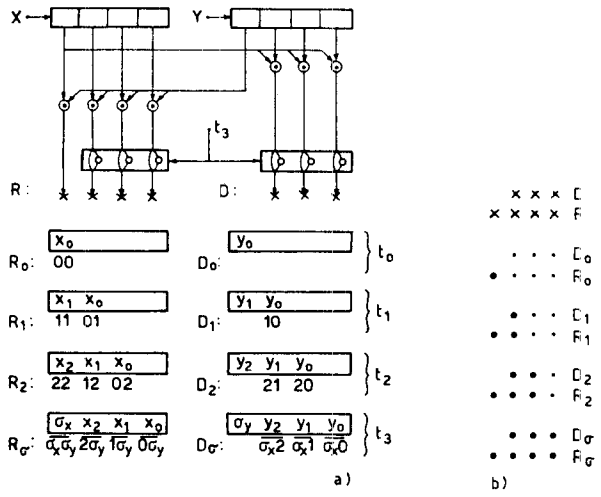

Fig.5 a): The logical scheme of a generator of diagonals and rows of fig. 3 array.
b): the dots represent the significant terms of the diagonals and of the rows at various clock times.

All the above variables are arranged so that those belonging to the same column have the same weight, the weight to the left of a given column being twice as much.

All input variables in any given column are used as inputs to a parallel counter, i.e. a combinatorial circuit whose purpose is to "count" the number of input variables whose value is "1", and to represent the count as a binary number at the outputs, [3, 7, 10]. A horizontal line separates the inputs variables from the outputs variables, lying below.

The outputs of a counter will belong to different columns: the least significant bit will lie on the same column to which belong the inputs, the second least significant bit will lie on the column to the left, etc. All outputs of each counter will be connected by a line, in order to signify that they are outputs of the same counter.

In fig. 7 it is shown how counters with more complex properties can be defined. For instance a 4-bit parallel adder can be considered as a counter, whose inputs have different weights (non-simple counter). As an additional convention, whenever an input variable is not used as input to a counter, it is reproduced as an isolated dot below the horizontal line. The scheme of fig. 6a (and the following) is build-up using the above convention and it can be described as follows. Starting from the two leftmost column, it is seen that they are composed by a single input, which is reproduced as output (without being used as input to a counter). The next column contains three variables, which are used as input to a (3;2) counter (i.e., a full adder). The third column contains six variables, that input to a (6;3) counter. The next column uses a (5;3) counter, and all the remaining a (2;2) counter each.

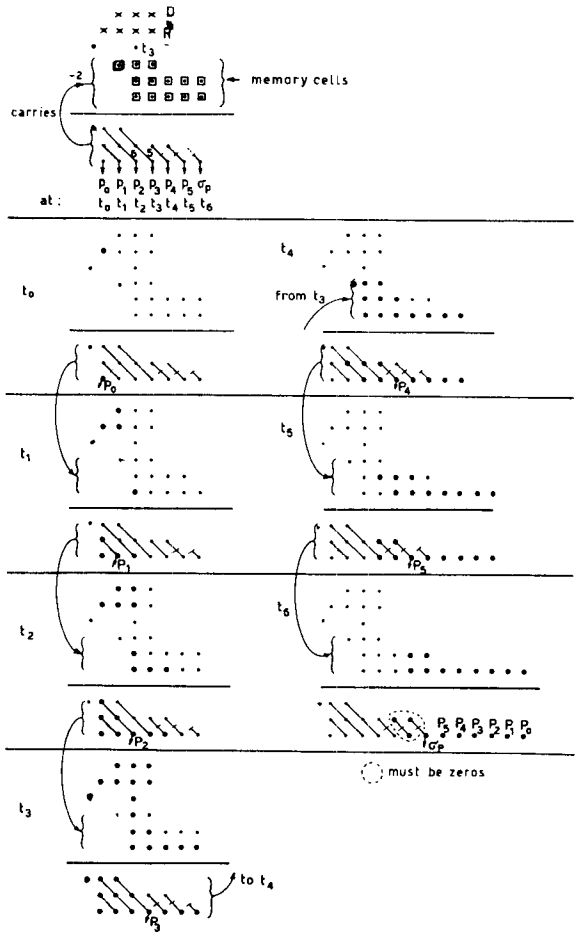

Fig.6 a): The logic scheme of the summer part of a multiplier, using the array generator of fig. 5. b): the operation of the logic scheme

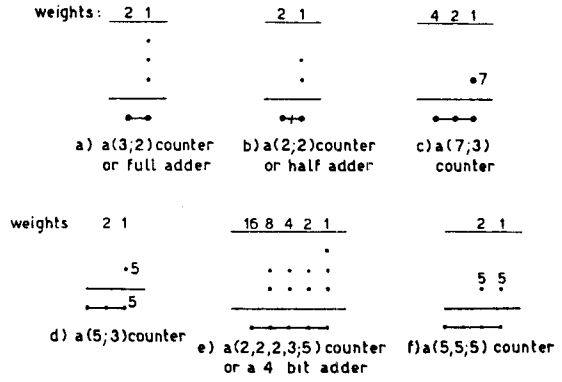

Fig.7 : Graphical representation for various types of parallel counters.

As shown in the figure, three output bits will belong to the leftmost column, three to the second; two to the next two columns, and a single bit to the remaining column.

The product bits, $p_0, p_1, p_2, \ldots$, appear at the corresponding clock times t_0, t_1, $t_2, \ldots$, at different outputs, as shown in the figure. (If a single output for P is desired, an additional (2n-1)-inputs OR will be required). Finally the output bits are fed back as carries as shown in the figure. Note that they are fed back shifted two position to the right: this accounts for the change of the next D and R's weight by a factor of 4, as required by the array generator (see preceding figure).

The operation of the above circuits at the times t_0, t_1, $t_2, \ldots$, is shown in fig. 6b, where the significant bits (i.e. those that can assume 0 or 1 values) are shown as dots. It can easily be verified that the above operation implements the scheme of fig. 3.

The array generator of a multiplier for positive factors is obtainable from the one necessary for two's-complement factors by omitting the bottom linear array of switches in fig. 5 (no information for D and R is required). Moreover, the t_3 inputs in fig. 6a are not required. In other words, the summer of the two's-complement multiplier is only slightly more complex than the one for positive factors, while its array generator requires 2n-1 switches for the transformation of D_σ and R_σ.

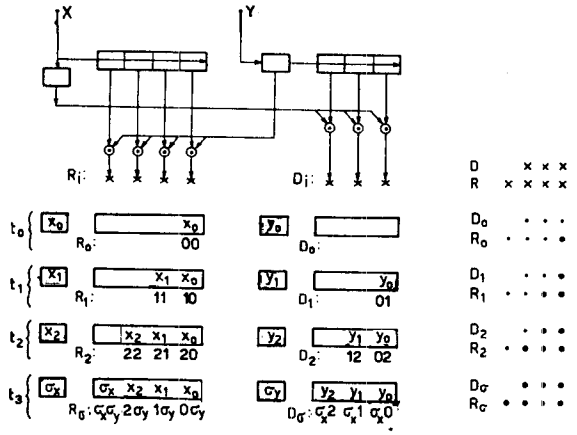

Fig.8 a): The logic scheme of a generator by diagonals and rows of fig. 2 array.
b): The dots represent the significant terms of the diagonals and of the rows at various clock times.

With an additional circuit, the most significant part of the product could be obtained in parallel, at time t_3: it is equivalent to the sum of all the carries from t_3 to t_4. This sum can be obtained by first reducing the three rows comprising those carries to an equivalent set of two rows (this can be done without carry propagation) and then adding these two rows in a parallel adder (with carry look-ahead circuits, if a greater speed is desired).

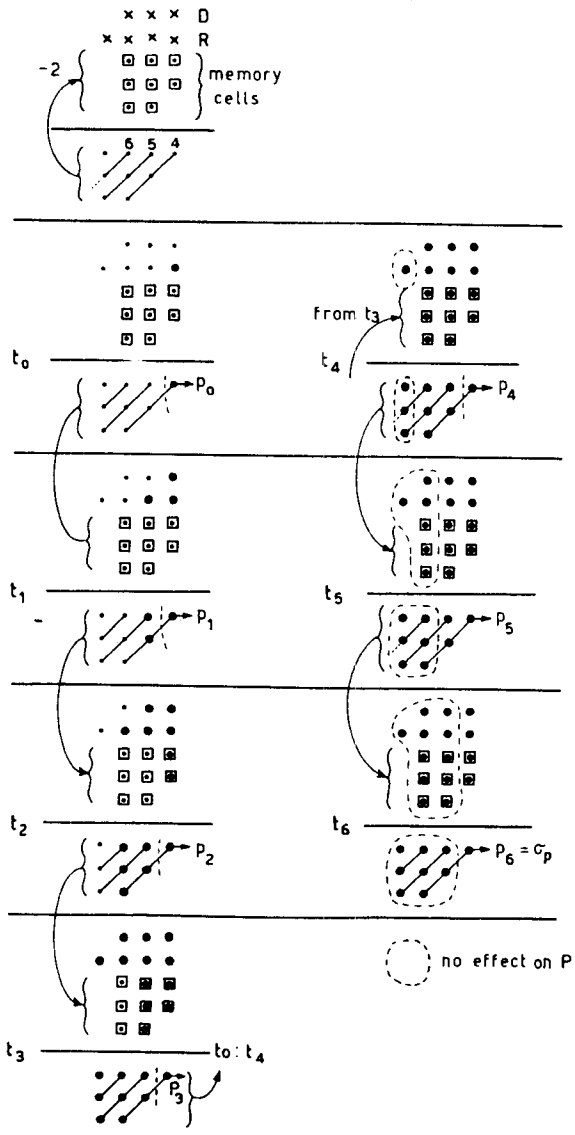

Fig. 9 a): The logic scheme of the summer part of a multiplier, using the array generator of fig. 8a). b): the operation of the logic schemes.

The scheme illustrated in fig. 6 is one of the several treated in [11] for positive factors: they can all be easily transformed for two's-complement factors following the rules here described.

For sake a brevity only a second scheme based on D and R generator will be illustrated, see fig. 8 and fig. 2.

It is composed by two stack registers (1), and two auxiliary memory cells, connected as shown in the figure, and an array of two-input AND gates. The operation of the circuit is shown also in figure 8a. In fig. 8b, the outputs composing D and R at the succeeding clock times are shown (compare

to fig. 5b). After time t_3, no changes will take place in the registers, so that D and R will be present at the outputs from t_3 to t_7. While in the scheme of fig. 5, the weights of D and R outputs, must be multiplied by four, in the scheme of fig. 8 the weights must be multiplied by two at each step: this can be verified by comparing fig. 8b with fig. 2. Fig. 9a represents the logical scheme of the summer, whose operation is represented in fig. 9b.

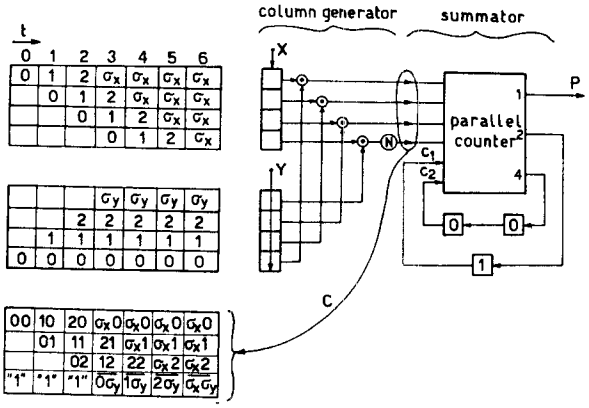

Fig.10 : The logic scheme of a multiplier based on an array generator producing the subsequent columns of the fig. 4 array. The summer is composed with a single parallel counter.

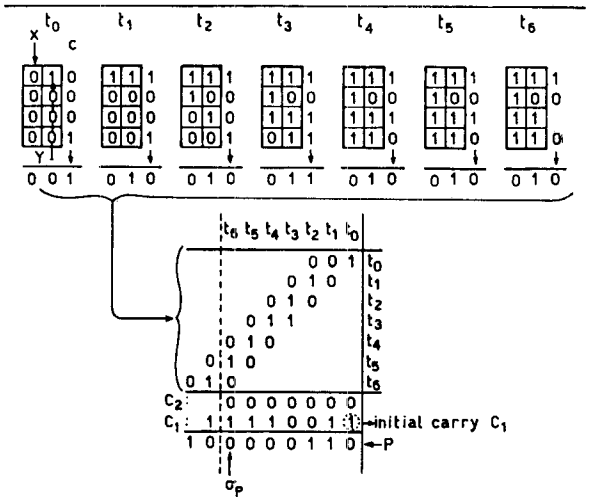

Fig.11 : The operation of the scheme of fig. 10, illustrated by the product of $X=1110_b$ (-2) and $Y=1101_b$ (-3).

This multiplier for two's-complement factor is exactly the same multiplier illustrated in [11] for positive factors. The only difference is in the operation of the circuit from t_4 to t_7: while for the case of positive factors D and R are zero from t_4 to t_7, in the multiplier for two's-complement factors, D and R must be mantained to the same value during t_3 to t_7.

We shall now consider the case of multipliers based on column generation.

A first scheme for such a multiplier is given in fig. 10.

The column generator is composed by a shift register (for X) and a stack register (for Y).

The same figure shows the operation of the column generator: it can be seen that the columns are generated according to the arithmetic scheme of fig. 4.

The summer part can take various forms (see [8]): one of them is represented in fig. 10: it comprises a 6 inputs parallel counter; 4 inputs are fed by the column generator, the remaining two by the feedback lines connected to the weight 4 and weight 2 counter outputs, through a 2-stage and a single stage register respectively. The product P is available at the weight 1 counter output. The only difference between the two's-complement multipliers of this type and the corresponding absolute value multipliers for the same number of factors bits is given by the negator placed at the bottom output of the column generator. Fig. 11 represents an example of multiplication with the above scheme.

Fig. 12a represents a second scheme for a column generator, composed by two shift registers X and Y. This scheme differs from the corresponding scheme for positive factors [11] for the extension of the two registers with the stages marked with an asterisk.

The operation of the circuits is illustrated in fig. 12b. The two registers Y and X are clocked alternatively, at times t'_1 and t''_1 respectively. The column terms will be zero until the first stages of both registers have been filled with some of the least significant bits: in fig. 12b, t''_1 represents the last clock time producing all zeros in the column generator. At the first clock t'_2 following t''_1, the content of both register is such (see fig. 12b, t'_2) that the term x_0y_0 is generated (i.e., the first column of the multiplier array). At the next clock, t''_2, the content of the registers is as shown in fig. 12b, t''_2 where it can be seen that the second column, comprising the terms x_0y_1 and x_1y_0 is generated; etc.

It can be easily verified that the scheme illustrated generates the multiplier array columns according to the arithmetic scheme of fig. 2. The summer necessary to obtain the product bits will be composed, as in fig. 10, by a parallel counter with a suitable number of inputs.

(1)Note the graphical symbols used for representing the two different types of registers: for the shift register the arrow from the input points to the first stage of the register, where the subsequent bits will be stored, while the previous bits are shifted ahead; in the stack register the arrow points to the last stage, where it will stay until the register is cleared, the second bit will be stored on top of it, etc.

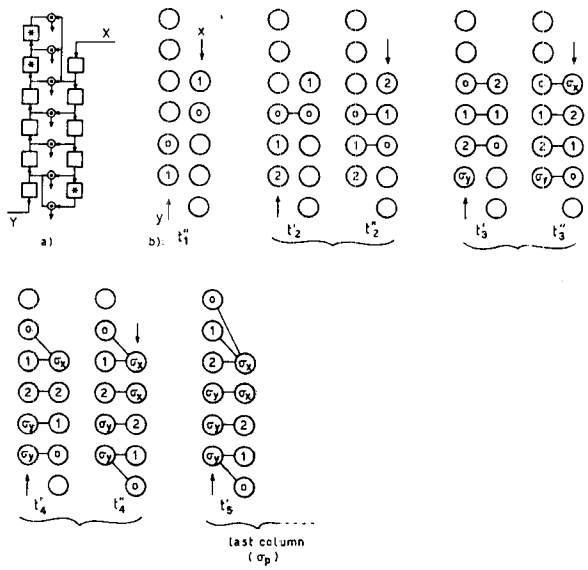

Fig.12 a): The logical scheme of a generator of the columns of fig. 2 array. b): the operation of the circuit.

CONCLUSIONS

The problem of designing multipliers for factors represented as serial binary numbers in two's-complement form has been considered. The proposed multiplier schemes have been derived from schemes illustrated in a previous paper for positive factors, and based on decomposing a multiplier into a first part, the array generator whose scope is to generate the terms of the multiplier array and a second part, the summer of the terms produced by the first part.

It has been shown how such schemes can be easily modified to handle numbers in two's-complement form.

REFERENCES

[1] L. Dadda, D. Ferrari: Digital multipliers: a unified approach, Alta Frequenza, vol. 37, pp. 1079-1086, Nov. 1968.

[2] E.E. Swartzlander: The quasi-serial multiplier, IEEE Trans. on Computers, vol. C-22, pp. 317-321, April 1973.

[3] E.E. Swartzlander: Parallel counters, IEEE Trans. on Computers, vol. C-22, pp. 1021-1024, Nov. 1973.

[4] C.R. Baugh, B.A. Wooley: A two's-complement parallel array multiplication algorithm, IEEE Trans. on Computers, vol. C-22, pp. 1045-1047, Dec. 1973 (with comments by D.E. Blankenship and D. Kroft, same journal, vol. C-23, pp. 1327-1328, Dec. 1974).

[5] J.A. Gibson, R.W. Gibbard: Synthesis and comparison of two's-complement parallel multipliers, IEEE Trans. on Computers, vol. C-24, pp. 1020-1027, Oct. 1975.

[6] T.G. McDonald, R.K. Guha, The two's-complement quasi-serial multiplier, IEEE Trans.on Computers, vol. C-24, pp. 1233-1235, Dec. 1975.

[7] W.J. Stenzel, et al.: A compact high-speed parallel multiplication scheme, IEEE Trans. on Computers, vol. C-26, pp. 948-957, Oct. 1977.

[8] L. Dadda: Multiple addition of binary serial numbers, IEEE Proc. 4th Symposium on Computer Arithmetic, pp. 14o-148, Santa Monica, Oct. 1978.

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

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

[11] L. Dadda: Some schemes for serial input multipliers, IEEE 6th Symp. Computer Arithmetic, Aarhus, pp. 52-59, June 1983.