SQUARERS FOR BINARY NUMBERS IN SERIAL FORM

Luigi DADDA

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

ABSTRACT

The problem of designing squarers for binary number in serial form (with the condition of the least possible delay between input and output) is treated.

Several schemes are illustrated, derived from fast multipliers for binary numbers in serial form, described in a previous paper.

It is shown that some of such multipliers offer a considerable saving in components when they are reduced to squarers. Some schemes are illustrated, both for positive and for two's-complement numbers.

INTRODUCTION

The problem of designing multipliers for binary numbers in serial form has been discussed in few papers. [2, 7, 10, 11]. In this paper the particular problem of multiplying a serial binary number by itself, i.e. of squaring a binary number, will be considered.

A squarer can, of course, be obtained from a regular multiplier, to whom the same number is applied as a multiplicand as well as a multiplier; but a regular multiplier used as a squarer will be redundant, since it is designed for two independent factors. It is therefore interesting to investigate on how a regular multiplier scheme could be simplified if it used as a squarer.

Such an investigation has been done, in the case of squarers for binary numbers in parallel form, by Chen [4] and Totadry Jayashree and Dhrube Basu [6], who were interested in using squarers for the synthesis of multipliers based on the quarter square algorithm.

In this paper, the case of squarers for binary numbers in serial form will be treated. It will be shown how the multipliers schemes illustrated in [10] can be easily reduced to squarers, some of them with a considerable saving in components.

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

GENERAL CONSIDERATIONS

The schemes for serial multipliers illustrated in [10] produce the products bits with the least possible delay with respect to the factors bits, i.e., each product bit is produced whenever its computation is logically possible with the available factor's bits.

The least significant product bit, y_0, can be generated immediately after the factor's least significant bits have been provided; the same holds for the following product and factor's bits.

As soon as the last factor's bits are applied, the last bit of the least significant half of the product can be produced. Moreover, the circuit contains all the information necessary to compute the most significant half of the product. It is therefore possible to conceive multipliers which produce in parallel the most significant half of the product as soon as the last factor's bits have been introduced.

Alternatively the most significant half of the product can be generated in serial form, following immediately the last bit of the product's least significant half.

The above considerations apply also to squarers, with the additional remark, see fig.1,

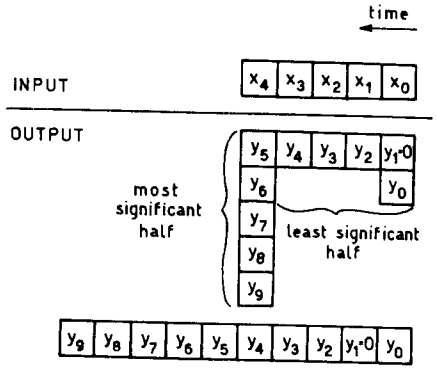

Fig.1 : To show the time relation between input and output

that the second least significant bit, y_1, is always zero (as it will be shown in the following with reference to fig. 2); y_2 can be computed as soon as x_1 has been provided, etc.

The serial multipliers described in [10] can be partitioned into two cascaded parts: the first, (the "array generator"), whose inputs are the two factor's bits streams (the least significant bits being the first), generates the elements of the multiplier array; the second part (the "summer") receives the latters and computes the product bits.

The array generator can be designed in various forms, depending on the way the array elements are produced to its outputs: all the array generator schemes comprise two registers, for storing the factor's bits, and a set of gates producing the multiplier-array elements.

The factor's registers can be conventional shift registers or stack registers (i.e. a set of binary cells in which the subsequent factor's bits are stored by means of a pointer, so that a given factor bit remains in the same cell once it has been written in). In some schemes both registers are shift-registers or stack-registers, some other schemes use one shift-register and one stack-register.

In using a multiplier as a squarer, both registers will contain the same number, but if both registers are of the same type only one will be needed, since the variables provided by the second register can be replaced by the variables stored in the first. In the case of registers of two different types this cannot be done, since a given input bit is in a fixed cell in the stack-register while it is moved from cell to cell in a shift-register.

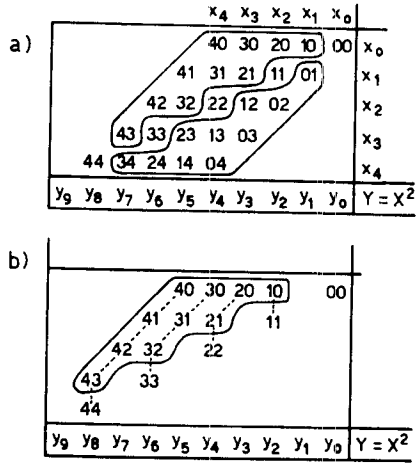

Fig.2 : a) The squarer array
b) The reduced squarer array

Consider now a multiplier array, see fig. 2a. In the case of two equal factors, i.e. in the case of a squarer: $x_i x_j = x_j x_i$ so that the part of the array below the anti-diagonal (composed by the terms x_i^2) is equivalent (i.e., it gives the same contribution to the product) to the part which is above the anti-diagonal.

The original array (fig. 2a) can thus be replaced, see [4], by a reduced, equivalent array, comprising the anti-diagonal and one of the above parts shifted by one position to the left, see fig. 2b (1)

As a consequence, the number of terms to be generated by the array-generator is smaller than in the case of a regular multiplier and a reduction in its complexity would then result. Moreover, the terms x_i^2 belonging to the anti diagonal reduce to x_i, since we are considering base-2 numbers.

In the following paragraph it will be shown that the above general remarks lead to various squarers schemes for positive integers.

In the last paragraph, the extension of the above schemes to the case of two's-complement numbers will be considered.

The notations used to describe the logic diagrams of the squarers are those adopted in [10, 11]. All the logic diagrams are based on the use of parallel counters [5,9].

SQUARERS FOR POSITIVE BINARY INTEGERS

Using the criteria exposed in the preceding paragraph, the various multiplier schemes given in [10] have been examined: in the following the most interesting schemes (i.e. those offering the least complexity) will be illustrated.

a) A group of schemes is based on the generation, at each step (i.e. whenever a new x_i is applied), of all the new array elements that can be computed using x_i and the operand bits already applied. In fig. 2b the squarer array elements that can be generated at each step are linked by a dotted line.

An array generator based on the above principle is shown in fig. 3. It is composed with an n stage shift register (n=5 in fig. 3), whose outputs feed an array of (n-1) 2-input AND gates. The figure shows the content of the register at the succeeding clock times, and the outputs of the circuit: it can be verified that these correspond, at each step, to the array terms which in fig. 2b are linked by a dotted line.

It can also be seen that, in order to have the correct weights, the initial weight of the leftmost output must be 2^0, and that all the weights must be multiplied by four at each step.

The outputs of the array generator are fed to the summer, in order to obtain X^2: the summer adds all the array terms generated in the succeeding steps.

More precisely, if S_{j-1} is the sum of all the elements of the sub-array generated at the (j-1)th step (S_{j-1} is therefore function of $x_0,\ldots,x_{j-1}$), then:

(1) It can be seen from fig. 2b) that y_1, the second least significant bit of X^2, is always zero.

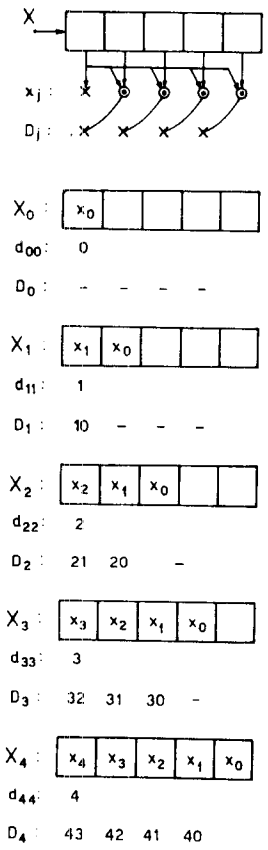

Fig.3 : A scheme for generating the reduced squarer array, "by diagonal" (see fig. 2b)

$$S_j = S_{j-1} + x_j + D_j ; \quad (j=0..n-1) \quad (1)$$
$$(S_{-1}=0)$$

For j=n-1 (last step) :

$$S_{n-1} = X^2$$

The above operation is performed by the summer. The logical scheme of a summer is given in fig. 4, where:

- the first two rows represent the variables produced by the array generator of fig. 3;
- the third row is a register where S_{j-1} is stored;
- the last row represents S_j, obtained by addition of the preceding rows.

It can be seen that the leftmost four columns contain two variables each: they feed a four stage parallel adder, whose outputs are represented by the five dots in the last row that are linked by a line. The rightmost five variables of the third row are needed only to store the least significant bits of X^2. The eight leftmost significant bits of the last row are fed back to the register (third row) after shifting them right by two positions: this shifting accounts for the multiplication by four

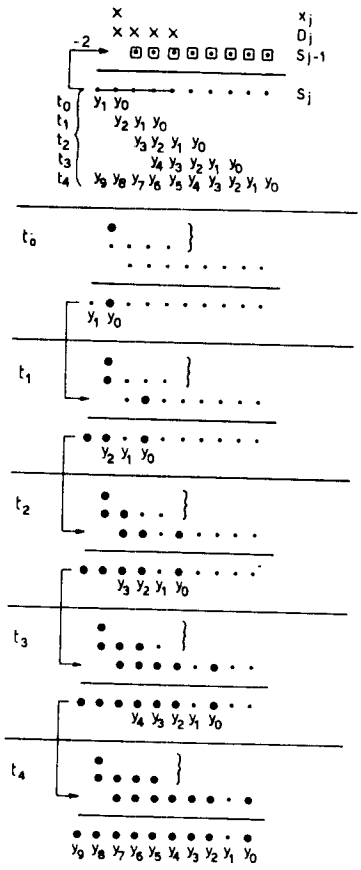

Fig.4 : a) Logical scheme of a summer with a carry propagation adder.
b) The operation of the scheme, at various clock times.

of the outputs of the array generator.

The operation of the logical scheme is represented in fig. 4b, where those variables that can be affected at the various steps are denoted by a dot, while a point is used for those variables that remain zero.

It can be seen that the squarer bits are generated at different outputs, the five least significant bits in the first five steps: at the last step, also the five most significant bits are generated, in parallel.

The speed of the scheme of fig. 4 is limited mainly by the adder (for which carry look-ahead circuits can of course be used).

If a greater speed is desired, the logical scheme of fig. 5 can be used: it has more memory elements, but it does not suffer the speed limitation of fig. 4 scheme, since no carry propagation occurs. Its operation is as follows. The first two rows represent the array outputs, as in the previous scheme. The third and fourth rows represent a set of two registers (composed by n-2 and 2n-2 cells respectively) whose sum is

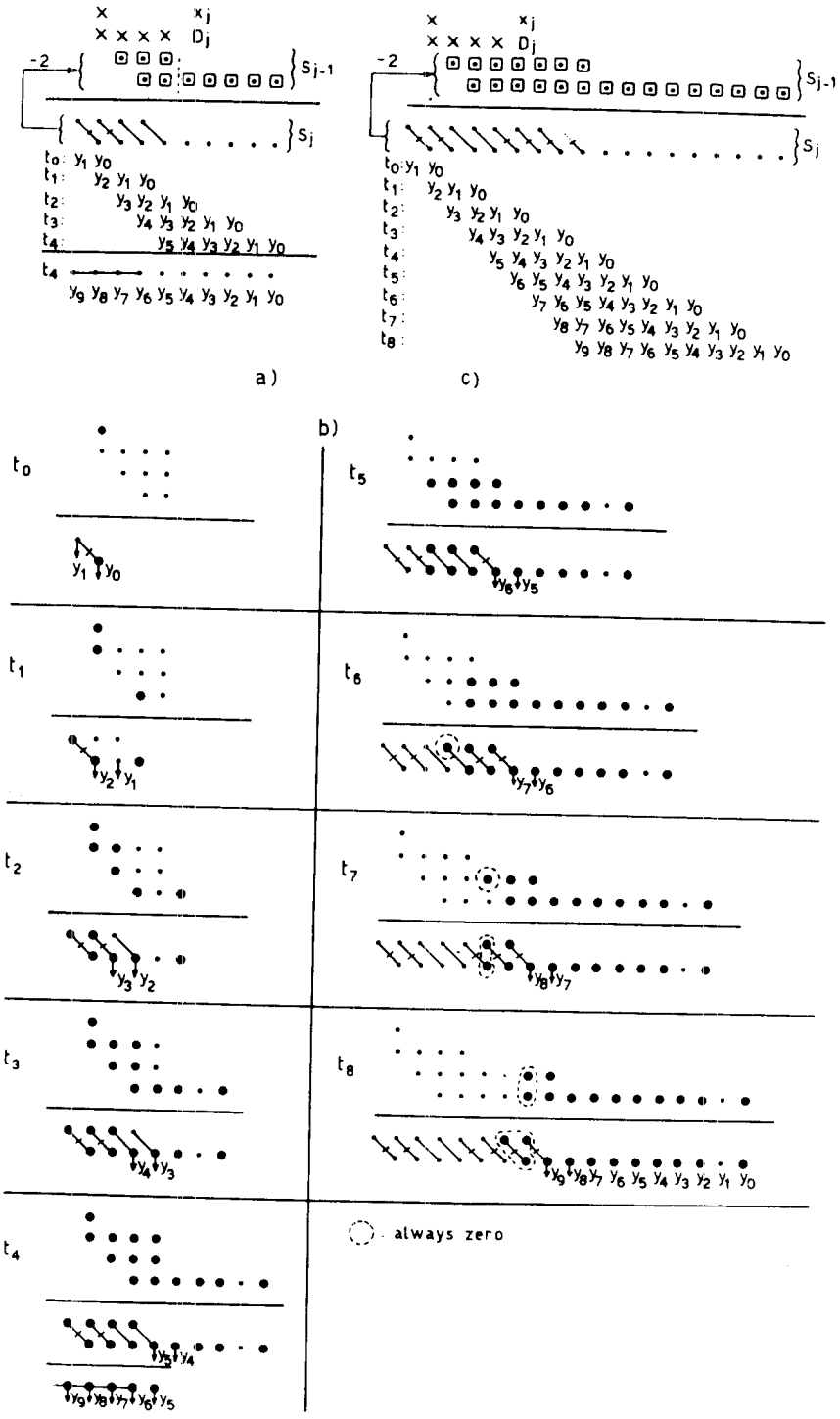

Fig. 5a): Logical scheme of a summer using an adder with no carry propagation.
b): The operation of the above circuit, at various clock times.
c): idem: the most significant half is generated serially.

equivalent, at each step, to the sum of all the array elements generated in the preceding steps.

This set of four rows can be reduced to two equivalent rows, using full-adders or half adders as shown in fig. 5a. The two rows thus obtained are fed-back, after shifting them right by two places, to the third and fourth rows.

The operation of the circuit is shown in fig. 5b. At clock times $t_0 \div t_4$ the square bits $y_0 \div y_4$ appears at different outputs, as shown (note that at t_0, both y_0 and y_1 are available, at t_1, y_1 and y_2, etc..).

At time t_4, also all the remaining most significant bits, $y_6 - y_9$ could be obtained in parallel with y_5, using an additional parallel adder.

Alternatively, the same bits can be obtained serially, with the circuit as in fig. 5c, whose operation is shown in fig. 5b, t_5 through t_8 (note that in t_6, t_7 and t_8, some bits shown as significant, i.e. represented by a dot, are in effect always zero: this is due to the fact that the value of the squarer array is represented redundantly by two rows instead of a single row as done in fig. 4).

In fig.5b and c) it is also shown that the circuit can be provided with additional memory elements (shown at the right of a vertical dotted line in fig. 5b) in order to store the complete result.

b) A second type of squarer scheme is shown in fig. 6, for the case n=5.

It is composed by an n bit shift-register and two sets of 2-input AND gates, generating the squarer array column by column. More precisely, two columns are generated simultaneously.

The operation of the circuit is shown in fig. 6b, where the content of the shift-register is shown at each step, t_0 through t_8 (compare to fig. 2).

In steps t_0 and t_1, no outputs are obtained (i.e., all outputs of the AND gates are zero).

At step t_2, the element '00' of the array is obtained; at step t_3, the elements 11 and 10 (i.e. the third column of the fig. 2 array) and the element 20 (i.e., the fourth column of the array) are obtained; at step t_4, the fifth column (22,21,30) and the sixth column (31,40) are obtained.

The remaining columns are produced in the same way (the last columns is produced at step t_6).

The columns thus generated can be handled by a suitable circuit (see [10]) which provides the result by generating two product bits at each of the steps t_2 through t_6. The above column generator can be easily extended for a larger n.

SQUARERS FOR TWO'S-COMPLEMENT NUMBERS

Being the square of a number always positive, no problems arise in the case of negative numbers

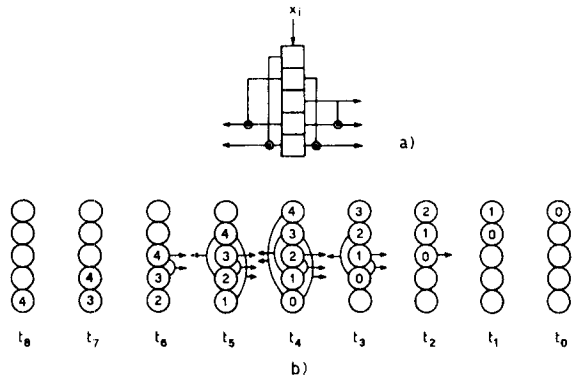

Fig.6 : a) Circuit for generating the squarer array by column (two columns at a time)
b) The operation of the above circuit, at various clock times

represented with sign and absolute value.

In the case of a negative number represented as two's-complement, one could change its sign using a simple circuit. In the case of a number to be squared, whose sign can be either positive or negative, it is desirable to have a squarer which can accept both positive and negative two's-complement numbers. The following well known properties of two's-complement numbers will be used in the following:

- the most significant bit of an n-bit number has a negative weight, $-2^{(n-1)}$; it is zero for positive numbers, one for negative numbers (it will be denoted in the following by x_σ, the sign bit)
- a two's-complement number can be extended to the left by repeating the sign bit x_σ:

  e.g. $X=x_\sigma x_2 x_1 x_0 = = \ldots 0\ 0\ x_\sigma x_\sigma x_\sigma x_2 x_1 x_0$

  All the x_σ have a weight corresponding to their position and the leftmost x_σ has a negative weight, according to the preceding property. In order to remind the negative weight of the leftmost sign bit, it will be marked by an asterisk

  $X=x^*_\sigma x_2 x_1 x_0 = \ldots 0\ x^*_\sigma x_\sigma x_\sigma x_2 x_1 x_0$
- In order to change the sign of a two's-complement number, complement all the bits and add a "one" into the least significant position

  -(0011) = 1100 + 1 = 1101 (-3)

Observe also that the largest negative value of an n bit two's-complement number is:

$X_{min} = -2^{(n-1)}$

(whereas $X_{max}=2^{n-1}-1$).

It can be easily shown that, if $-2^{(n-1)}$ is included in the range of X, then X^2 will require 2n-1 bits; if it is excluded from the range of X (i.e. if $X_{min}=-2^{(n-1)}+1$) then X^2 will require 2n-2

bits).

In the following, both cases will be considered.

Two's-complement numbers can be squared following two different approaches.

In the first approach, the sign x_σ of the operand X is extended to the left at least n times, and the squarer array is then constructed, see fig. 7.

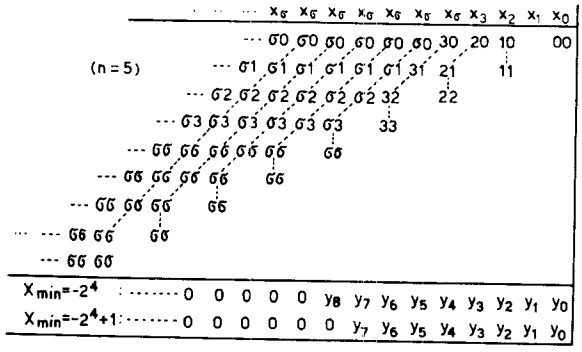

Fig.7 : A squarer array for two's-complement numbers, in which the sign bit has been extended indefinitely

The square $Y=X^2$ can then be computed in the usual way by computing the value of the squarer-array.

The squarer-array elements can be generated either "by diagonal" (see fig. 7) or "by column", as shown in the preceding paragraph.

It can be easily verified that, for the same n, the complexity of the circuits for a two's-complement squarer is about twice the complexity of a positive integer squarer.

A better solution can be achieved by observing in fig. 7 that after x_σ has been introduced, the same diagonal (comprising the terms $\sigma 3$, $\sigma 2$, $\sigma 1$, $\sigma 0$) must be added in three more steps ($t_5..t_7$) in order to obtain the result. Moreover, the same diagonal must be shifted left one place at each step: this can be obtained in the logical scheme of fig.8 by shifting S_j one place left (instead of two), from t_5 to t_7.

At t_4, the sign bit x_σ has been applied, and the bit y_5 generated. At t_5, the outputs from the array generator remain unchanged, but S_4 is transferred to the register S_{j-1} by shifting it right one place only.

The same is done in t_6, and, if $X_{min}=-2^4$, also in t_7. Note also that, only part of terms belonging to the diagonal $\sigma 3$, $\sigma 2$, $\sigma 1$, $\sigma 0$ affect the final result: those terms which do not affect the result are circled with a dotted line (pseudo-significant bits) for $X_{min}=-2^4$ and with a continuous line for $X_{min}=-2^4+1$.

A second approach to the design of squarers for two's-complement numbers is based on the squarer array constructed by assuming that the weight of the most significant bit is negative: see fig. 9a where the symbol σ^* for the most significant bit has been marked with an asterisk for denoting that it has a negative weight. The product can be obtained by computing the value of the array, taking into account the signs of the elements (note that $\sigma^* \cdot \sigma$ is zero or a positive one).

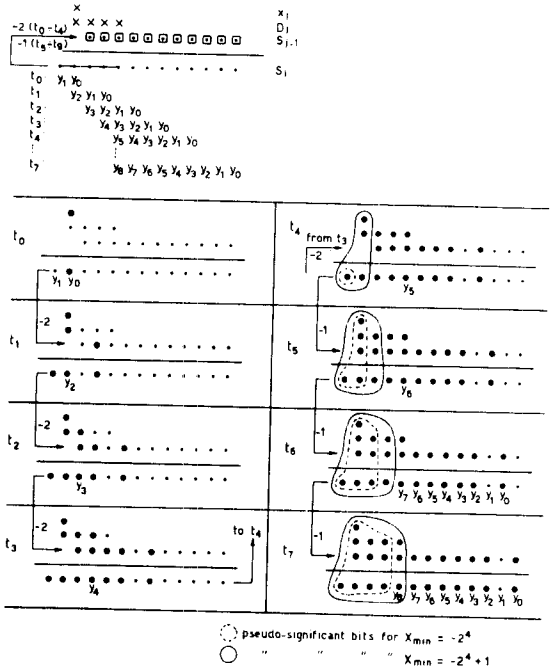

Fig.8 : The logical scheme of a squarer for two's-complement numbers, fed by the array-generator of fig. 3

The array can be transformed into an equivalent one, comprising only zeros or positive elements, see fig. 9b: it is obtained from fig. 9a array by replacing its leftmost diagonal ($\sigma^* 3$, $\sigma^* 2$, $\sigma^* 1$, $\sigma^* 0$) by a diagonal composed by the complements ($\overline{\sigma^* 3}$, $\overline{\sigma^* 2}$, $\overline{\sigma^* 1}$, $\overline{\sigma^* 0}$), and adding a "1" into the least significant column (corresponding to $\overline{\sigma^* 0}$).

Such an array can be generated by diagonals with the circuit of fig. 10a, which is obtained from the circuit of fig. 3 by providing at the outputs of the register X an array of gates, capable of producing the squarer array elements by diagonals, and, at time t_4 the complements $\overline{\sigma 3}$, $\overline{\sigma 2}$, $\overline{\sigma 1}$, $\overline{\sigma 0}$ of the last diagonal. The "1" needed in the array of fig. 9b is implemented by providing, at time t_4, an auxiliary input, as shown in fig. 10a. The subsequent partial sum of the array is stored in the register S_{j-1}.

All the variables lying above the horizontal line are transformed into a single row of variables: no processing is required for the five rightmost variables, while those belonging to the five leftmost columns are inputs to a five stage parallel adder (whose outputs are linked by a line). The six leftmost bits of S_j are fed back in the S_{j-1} register, after shifting them right two places.

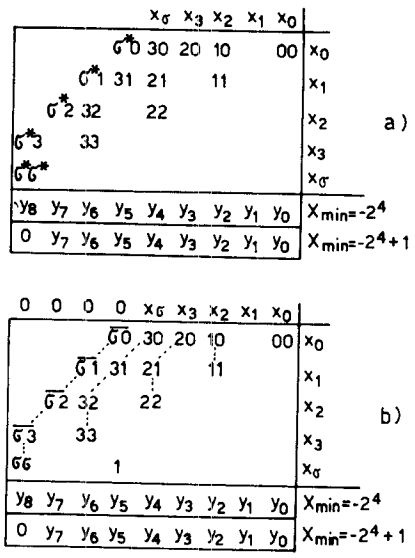

Fig.9 : a) A squarer array for two's-complement numbers
b) An equivalent squarer array with only positive elements

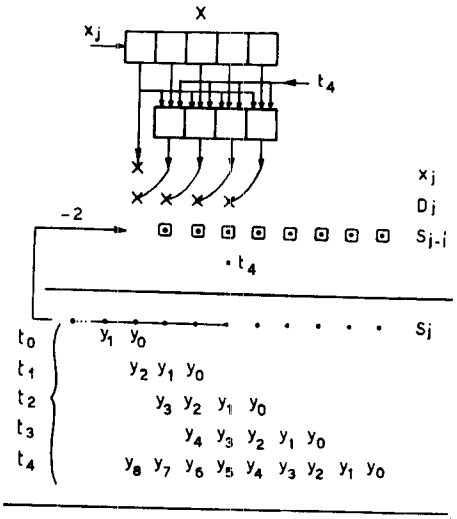

Fig.10 : a) A squarer for two's-complement numbers, composed by: an array generator "by diagonal" and a summer using a carry propagation adder

The same figure shows the times at which the square's bits y_i are generated and the corresponding outputs. Fig. 10b illustrates the operation of the circuit.

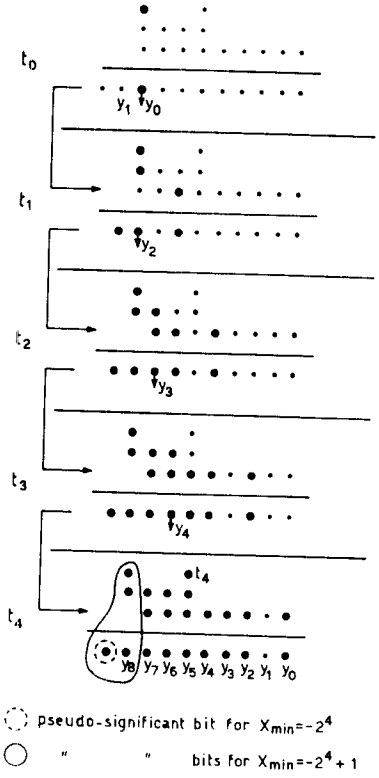

Fig.10 : b) The operation of the above circuit at various clock times

CONCLUSIONS

Squarers for binary numbers in serial form, capable of providing the result with the least possible delay between input and output, can be designed according to a scheme already used by the author for designing serial multipliers. Such a scheme is composed with an array generator and a summer. The array generator produces the elements of the squarer array as soon as new bits are applied to the inputs: the array elements produced can be generated by "diagonals" or by "columns". The summer accumulates the array elements thus generated, and produces the bits of the square. Various schemes are illustrated, some of them providing the most significant half of the square in parallel, some other in serial form. The case of squarers for negative numbers represented as two's-complements has also been considered.

REFERENCES

[1] L. Dadda: Some schemes for parallel multipliers, Alta Frequenza, vol. 34, pp. 349-356, May 1965.

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

[3] H. Ling: High-speed computer multiplication using a multiple-bit decoding algorithm, IEEE

Trans. Comput., vol. C-19, pp. 706-709, Aug. 1970

[4] T.C. Chen: A binary multiplication scheme based on squaring, IEEE Trans. Comput., vol. C-20, pp. 678-680, June 1971.

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

[6] Totadri Jayashree, Dhruba Basu: On binary multiplication using the quarter square algorithm, IEEE Trans. Comput., vol. C-23, pp. 957-960, Sept. 1976.

[7] J. Kane: A low-power, bipolar, two's complement serial pipeline multiplier chip, IEEE Journal Solid-state Circuits, vol. SC-11, pp. 669-678, Sept. 1976.

[8] D.P. Agrawal: A novel technique for computing negabinary squares, IEEE Trans. Comput., vol. C-7, pp. 266-270, March 1978.

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

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

[11] L. Dadda: Fast multipliers for two's-complement numbers in serial form, IEEE Symp. Computer Arithmetic, Urbana, Ill., June 1985.