A Recursive Algorithm for Binary Multiplication and Its Implementation

RENATO DE MORI and RÉGIS CARDIN
Concordia University

A new recursive algorithm for deriving the layout of parallel multipliers is presented. Based on this algorithm, a network for performing multiplications of two's complement numbers is proposed. The network can be implemented in a synchronous or an asynchronous way. If the factors to be multiplied have N bits, the area complexity of the network is $O(N^2)$ for practical values of N as in the case of cellular multipliers. Due to the design approach based on a recursive algorithm, a time complexity $O(\log N)$ is achieved.

It is shown how the structure can be pipelined with period complexity $O(1)$ and used for single and double precision multiplication.

Categories and Subject Descriptors: B.2.1 [**Arithmetic and Logic Structures**]: Design Styles—*parallel; pipeline*; C.5.4 [**Computer Systems Organization**]: Computer System Implementation—*VLSI*

General Terms: Algorithm, Design, Performance

Additional Key Words and Phrases: Multiplier, layout, complexity

1. INTRODUCTION

Multipliers are fundamental components of computer arithmetic units and signal processing systems. In the last thirty years, the design of parallel multipliers has received considerable attention. A fundamental contribution to the design of combination or simultaneous multipliers has been given by Wallace [21]. He proposed a network of Carry Save Adders (CSA) for adding Partial Products (PP) generated by two integer factors represented in binary code with N bits each and obtaining in $O(\log N)$ time two addends whose sum is equal to the product of the factors. A pipelined version of this design has been implemented in commercially available machines. Wallace's design approach was improved by Dadda [5] who proposed to use Parallel Counters (PC) instead of CSA in order to reduce cost.

In the late 1960s, a number of multiplier designs were proposed based on iterative arrays of equal cells [3, 6, 7]. These structures have an area complexity $O(N^2)$ and time complexity $O(N)$. The basic cell of the iterative multipliers was a gated full adder. *Cellularity* was considered an advantage for Large Scale Integration even if the time complexity of cellular structures was $O(N)$ rather than $O(\log N)$. In fact, their layout can be automatically generated by iteratively

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Formation Chercheurs Action Concertée of the provincial government of Quebec.
Authors' address: Department of Computer Science, Concordia University, 1455 De Maisonneuve Blvd. West, Montreal, Quebec, Canada, H3G 1M8.
This paper is dedicated to the memory of Professor Rinaldo Sartori.

reproducing the layout of a single cell on a plane. Using the same cells, networks for multiplying signed numbers were proposed [1, 9, 17] possessing the same area and time complexity. An extended review with interesting comments and contributions to the design of parallel multipliers can be found in the book by Hwang [13].

In order to increase the speed of iterative multipliers with time complexity $O(N)$, some macrocellular structures have also been proposed. A new solution for the macrocell designs based on multiplexers has been proposed recently [8]. It allows for the reduction of the maximum delay of the multiplier array to a *fraction* of N.

A recent paper by Capello and Steiglitz [4] proposes a VLSI layout for parallel multipliers with an area complexity $A = O(N^2 \log N)$ and a time complexity $T = O(\log N)$. Capello and Steiglitz compare the existing solutions on the basis of a VLSI Figure of Merit (FM) defined as follows:

$$\mathrm{FM} = \mathrm{AT}^2(\mathrm{PE})^2 \tag{1}$$

where A is the area complexity, T is the time complexity, and PE is the period complexity.

Capello and Steiglitz [4] have shown that their multipliers can be pipelined with a period complexity $\mathrm{PE} = O(1)$ corresponding to a figure of merit:

$$\mathrm{FM}_1 = N^2 \log^3 N. \tag{2}$$

They also derived a Lower Bound of Figure of Merit (1) for parallel multipliers (LBFM):

$$\mathrm{LBFM} = N^2 \log^2 N \tag{3}$$

and reported that this lower bound was not reached by any solution published before their paper.

A new recursive algorithm for the layout generation of parallel multipliers is presented in this paper. The area complexity of multipliers obtained by the use of such algorithm is $O(N^2)$ for values of N not exceeding a threshold greater than 100, while the time complexity is $O(\log N)$. The implementation of such a scheme requires essentially two types of cells: carry-save adders and multiplexers. Positive as well as two's complement numbers can be handled. The structure can be pipelined with a period complexity $O(1)$ thus reaching the lower bound (3) of FM as defined by (1).

Section 2 of this paper introduces the basic algorithms. Section 3 discusses area and time complexities. Section 4 deals with multiplication of two's complement numbers, an issue that is rarely discussed in the presentations of binary multipliers with logarithmic time complexity. Section 5 shows how recursive multipliers can be pipelined. Application and conclusions are presented in Section 6.

2. THE ALGORITHM FOR RECURSIVE MULTIPLIERS

In order to introduce the algorithm for designing recursive multipliers, an iterative algorithm, denoted Algorithm IM (Iterative Multiplication), for multiplying positive binary integer numbers will be first presented. This algorithm leads to the implementation of cellular multipliers with area complexity A =

$O(N^2)$ and time complexity $T = O(N)$. A design for a basic cell of these multipliers is proposed in [8]. These cells reduce the time complexity to a fraction of N allowing one to design fast multipliers for small values of N. Furthermore, these cellular multipliers are implemented by repeating an elementary cellular structure which is a big advantage from the point of view of automatic design, manufacturing, and testing. Iterative multipliers, also called *array multipliers* implemented as proposed in [8] are testable [11] and can be pipelined with a period complexity $O(1)$.

2.1 Algorithm IM

Let

$$H = \sum_{i=0}^{N-1} h_i 2^i \tag{4}$$

and

$$K = \sum_{j=0}^{N-1} k_j 2^j \tag{5}$$

be positive integer factors and

$$P = H*K = \sum_{r=0}^{2N-1} p_r 2^r \tag{6}$$

be the product. h_i, k_j, p_r are binary variables.

Let $N = RG$ with R and G integers. Wording the factors H and K in the basis 2^G one gets:

$$K = \sum_{n=0}^{R-1} K_n 2^{nG} \qquad H = \sum_{m=0}^{R-1} H_m 2^{mG} \tag{7}$$

with H_m and K_n integers less than 2^G and expressible in the binary code with G bits.

Figure 1 shows the structure of the iterative multiplier for the case $G = 2$ and $N = 6$. Some free inputs that can be used for adding two numbers are:

$$V = \sum_{i=0}^{N-1} v_i \qquad \text{and} \qquad W = \sum_{j=0}^{N-1} w_j.$$

The input wires where V and W are applied are called *additive input* wires. The structure shown in Figure 1 is thus capable of performing the operation

$$Z = H*K + V + W, \tag{8}$$

which is also the expression computed by the basic cell. Dean [6] has called such a structure, a *full-multiplier.*

A special cell design for $G = 2$ is proposed in [8]. This cell design is based on four 16 input–1 output multiplexers where variables affected with the highest delay are applied at the *select* input. In this way the cell has a propagation delay equal to the switching time of a multiplexer.

The area complexity in this case is

$$A = O(N/2)^2 = O(N^2). \tag{9}$$

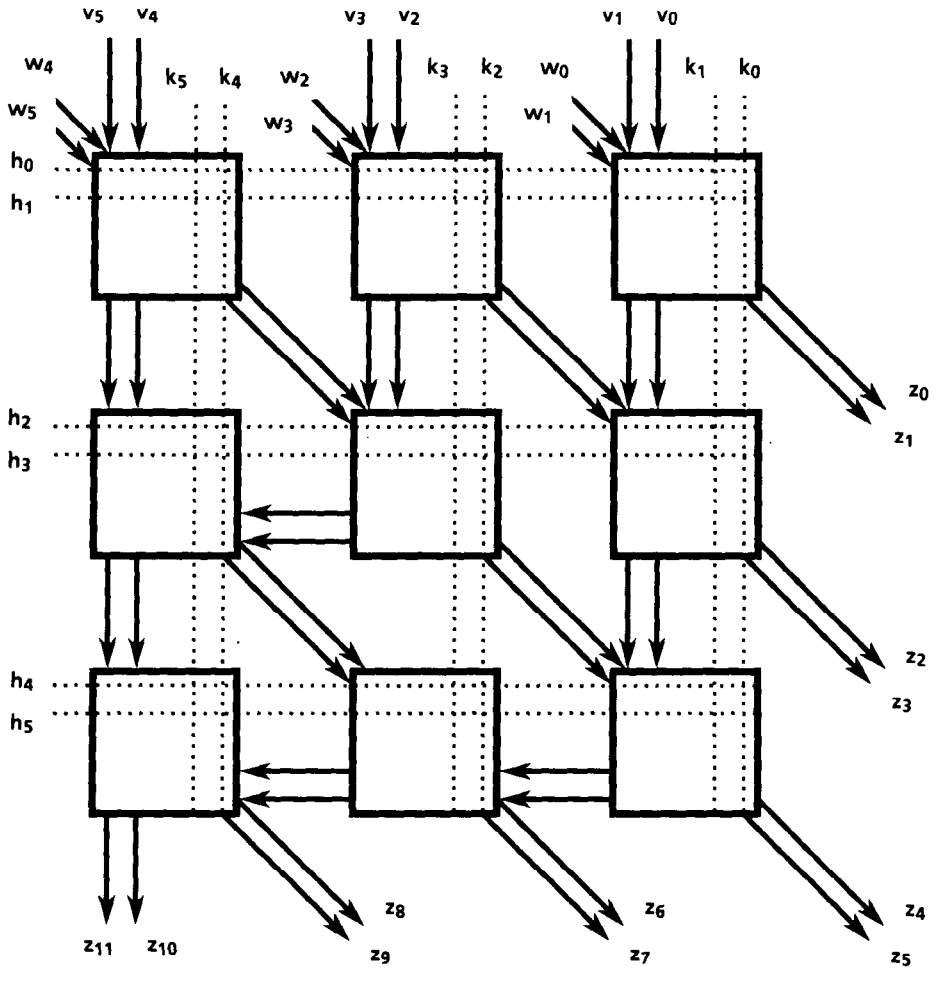


Fig. 1. Iterative multiplier.

Let T_m be the switching time of a multiplexer; the structure has a total multiplication delay of

$$T_d = NT_m = O(N). \tag{10}$$

Cells with $G > 2$ can be designed following the approach proposed in [8] with multiplexers having $2G$ select inputs. This would reduce the multiplication delay to $(2N/G)T_m$.

An algorithm has been proposed [9] for using full multipliers of positive numbers in order to multiply two's complement binary fractions with the addition of peripheral logic which does not affect the area, time, or period complexity expressions. Another algorithm has been proposed by Baugh and Wooley for two's complement binary multiplication [1]. It has been shown that the structure of Figure 1 can be adapted to perform multiplications of two's complement numbers with Baugh and Wooley's algorithm [8]. Testability of two's complement multipliers based on iterative arrays of multiplexers has been investigated [11].

In order to reduce the time complexity from $O(N)$ to $O(\log N)$ by keeping the area complexity approximately $O(N^2)$, a new structure, called *recursive multiplier* is proposed in this paper. This structure shares the following features with the iterative multiplier proposed in [8]:

—modular layout,
—possibility of pipelining and multiplying two's complement factors,
—possibility of using the structure for a single double precision multiplier or for four single precision multipliers.

An algorithm for designing recursive multipliers and for generating their layout will be given in the following section.

2.2 Algorithm RM

The Recursive Multiplication algorithm (RM) will be introduced using the same assumptions and notations as in relations (4), (5), (6) and (7) of the IM algorithm.

For every step of the algorithm, the basic operations performed concurrently will be given. A device performing each basic operation will be defined and the details on how the device output is represented will be indicated.

Every output, represented by a capital letter, is supposed to be in pure binary code (1 bit per weight). The first step of the algorithm, *step 0*, consists of partitioning the N-bits of K and of H into adjacent G-tuples and computing in parallel all the products between K_n $(1 \leq n \leq R)$ and H_m $(1 \leq m \leq R)$.

This product will be represented as

$$P^0_{m,n} = H_m K_n 2^{(m+n)G},$$

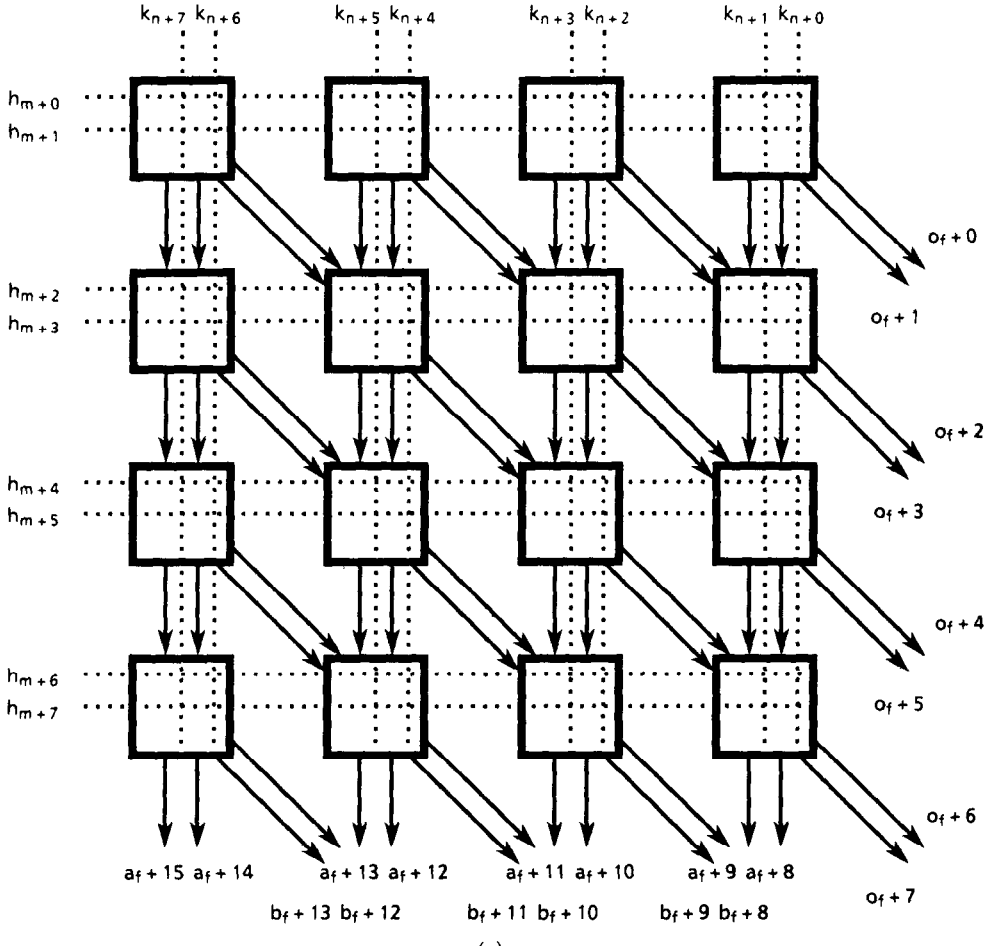
where the superscript 0 remembers that the product is the result of the computation performed at step 0. Products $P^0_{m,n}$ can all be computed with iterative arrays of macrocells based on multiplexers. The delay introduced by these arrays can be reduced by allowing the most significant bits of the output to be 2 for each weight.

A layout for such arrays is shown in Figure 2a. There is one output for the first G least significant bits. Let $L^0_{m,n}$ be the binary number represented by these bits. On the contrary, there are two outputs per weight for the G most significant bits. These output bits can be considered as forming two binary numbers,

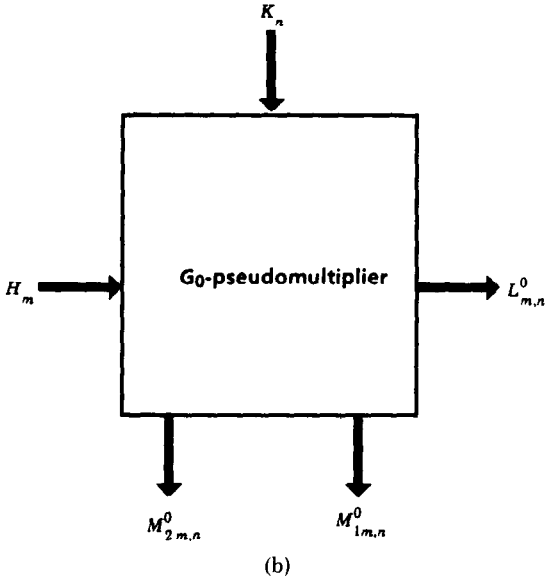
$$M^0_{1m,n} \quad \text{and} \quad M^0_{2m,n},$$

where each of these numbers has one bit per weight.

step 0 $(0 \leq m < R,\ 0 \leq n < R)$
for every pair (m, n) **do**
cobegin
:
:
Compute: $P^0_{m,n} = H_m K_n 2^{(m+n)G}$
:
:
coend



(a)



(b)

Fig. 2. (a) Iterative G_0-pseudomultiplier (detailed scheme). (b) Iterative G_0-pseudomultiplier.

Each concurrent operation is performed by a device called a **G_0-pseudomultiplier** whose input-output characteristics are defined as follows:

G_0-pseudomultiplier (m, n)

Input: H_m, K_n

Output: $P^0_{m,n} = H_m K_n 2^{(m+n)G}$

The output is represented as follows:

$$P^0_{m,n} = L^0_{m,n} 2^{(m+n)G} + (M^0_{1m,n} + M^0_{2m,n}) 2^{(m+n+1)G}$$

$$\cdot\ (L^0_{m,n} < 2^G,\ M^0_{1m,n} < 2^G,\ M^0_{2m,n} < 2^G).$$

Figure 2a shows the layout of a G_0-pseudomultiplier with $G = 8$ using the cells whose design has been introduced in [8]. The area complexity of this multiplier is

$$A_G = (G/2)^2 \tag{11}$$

and the time complexity is

$$T_G = GT_m. \tag{12}$$

Figure 2b shows a schematic representation of a G_0-pseudomultiplier.

Each cell in Figure 2a is, a 2 bits **full multiplier**. In the figure the following assumptions have been made:

$$K_n = \sum_{i=0}^{7} k_{n+i} 2^i,$$

$$H_m = \sum_{i=0}^{7} h_{m+i} 2^i,$$

$$L^0_{m,n} = \sum_{i=0}^{7} o_{m+n+i} 2^i,$$

$$M^0_{1m,n} = \sum_{i=0}^{7} a_{m+n+i+8} 2^i,$$

$$M^0_{2m,n} = \sum_{i=0}^{5} b_{m+n+i+8} 2^i.$$

Step 1 of the algorithm consists in grouping G_0-pseudomultipliers into squares of four each and in pseudoadding the outputs using carry-save pseudoadders. Pseudoadditions are performed on six numbers and produce two numbers whose sum is equal to the sum of the initial six. Pseudoaddition is a carry-save operation performed in a time that does not depend on the number of bits of the addends. The operation is repeated until two numbers whose sum is equal to the product are found.

Let **step y** be the generic pseudoaddition step and let **step Y** be the final step.

Step y

Let: $n_y = 0, 2^y, \ldots, q2^y, \ldots, R - 2^y;$

$m_y = 0, 2^y, \ldots, p2^y, \ldots, R - 2^y;$

for every (m_y, n_y) **do**

cobegin

:

:

Compute: $P^y_{m_y,n_y} = P^{y-1}_{m_y,n_y} + P^{y-1}_{m_y+2^{y-1},n_y} + P^{y-1}_{m_y,n_y2^{y-1}} + P^{y-1}_{m_y+2^{y-1},n_y+2^{y-1}}$

:

:

coend

Each concurrent operation is performed by a device called a **G_y-pseudomultiplier** whose input-output characteristics are defined as follows:

G_y-pseudomultiplier (m_y, n_y)

input: $$H_{m_y} = \sum_{m=m_y}^{m_y+2^y} H_m 2^{mG}$$

$$K_{n_y} = \sum_{n=n_y}^{n_y+2^y} K_n 2^{nG}$$

output: $$P^y_{m_y,n_y} = H_{m_y} K_{n_y} 2^{(m_y+n_y)G}$$

The output is represented as follows:

$$P^y_{m_y,n_y} = (L^y_{1m_y,n_y} + L^y_{2m_y,n_y})2^{(m_y+n_y)G} + (M^y_{1m_y,n_y} + M^y_{2m_y,n_y})2^{(m_y+n_y+2^y)G}. \quad (13)$$

The computation of two numbers

$$\begin{aligned} P^Y_1 &= (L^Y_1 + M^Y_1)2^{2RG} \\ P^Y_2 &= (L^Y_2 + M^Y_2)2^{2RG}, \end{aligned} \quad (14)$$

such that

$$P = P^Y_1 + P^Y_2$$

is obtained by performing step #0 and then repeat step y until $y = Y$ such that

$$2^Y = R. \quad (15)$$

There will be only one pseudomultiplier G_Y-(0, 0) giving two output bits for each weight. These outputs can be added using a special adder having $O(\log N)$ time complexity [2].

Figure 3 shows an example of the execution of a multiplication using the algorithm introduced so far. Numbers are represented in decimal code with one digit per weight for the sake of simplicity.

Although the algorithm proposed so far is iterative, a recursive algorithm can be introduced for defining a G_Y-pseudomultiplier in terms of G_Y-pseudomultipliers

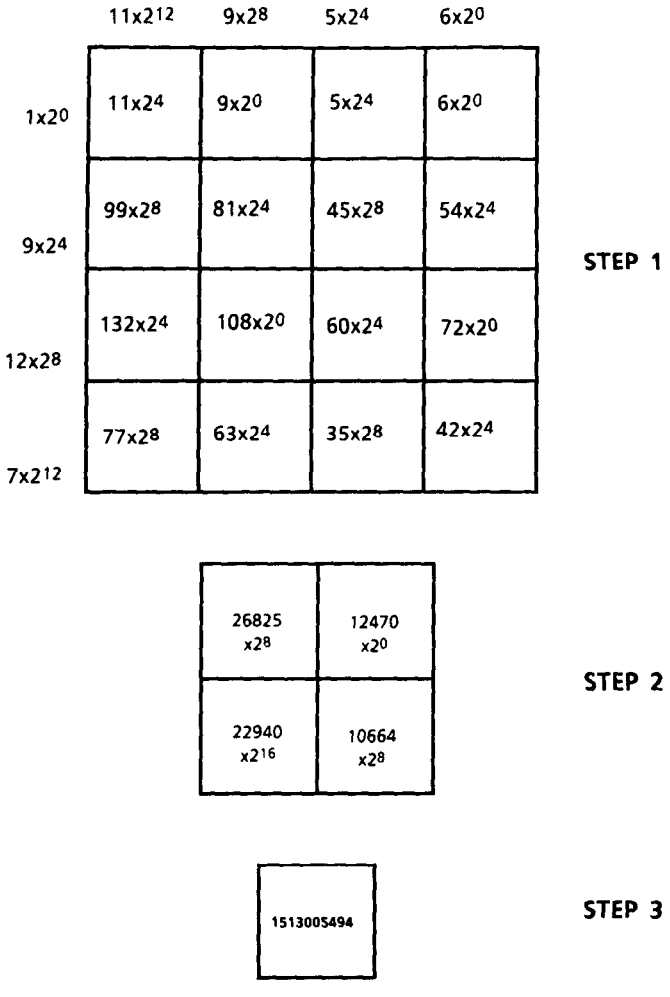


Fig. 3. Recursive multiplication example.

$(0 \leq y < Y)$. The recursive algorithm can be used for the generation of the multiplier layout.

G_y-pseudomultiplier (m, n)
begin
  if $y = 0$ **then**
    compute $P^0_{m,n}$ using iterative arrays
  else
  cobegin
    $G_{(y-1)}$-pseudomultiplier (m, n);
    $G_{(y-1)}$-pseudomultiplier $(m + 2^{y-1}, n)$;
    $G_{(y-1)}$-pseudomultiplier $(m, n + 2^{y-1})$;
    $G_{(y-1)}$-pseudomultiplier $(m + 2^{y-1}, n + 2^{y-1})$;
  coend

compute $P^y_{m,n}$ as represented in **step y** using Pseudo-Adders (PA) and represent it by two numbers

$$P^y_{1m,n} = (L^y_{1m,n} + L^y_{2m,n})2^{(m+n)G}$$

$$P^y_{2m,n} = (M^y_{1m,n} + M^y_{2m,n})2^{(m+n+2^y)G}$$

end

Notice that the size of a G_y-pseudomultiplier (defined as the number of factor bits involved in the operation) is 2^yG. Notice also that with the notation adopted here the index of the recursion is the subscript of G.

The algorithm of the entire multiplier can be described as

multiplier (H, K);
inputs: H, K;
output: P;
begin
  $y := Y$;
  G_y-pseudomultiplier (0, 0);
  $P := P^Y_1 + P^Y_2$;
end;

The addition of P^Y_1 and P^Y_2 that represents the output P^Y of the G_Y-pseudomultiplier is performed by a Special Adder. The way $P^y_{m,n}$ is computed using the outputs of 4 $G_{(y-1)}$-pseudomultipliers is shown in Figure 4. PA stays for **Pseudo-Adder**. Symbols in Figure 4 have been simplified for the sake of clarity. Some auxiliary variables OL_{ij} and OM_{ij} have been introduced ($1 \le i, j \le 2$) for representing pairs of numbers that are the results of partial pseudoadditions performed by PAs.

Notice that here pseudoadditions reduce, with a carry-save operation, six binary numbers to two binary numbers whose sum is equal to that of the six addends. The details of the operation performed by the PAs in Figure 4 are given in the following:

$$(OL^{y-1}_{21} + OL^{y-1}_{22})2^{(m+n+2^{y-1}G)} = (M^{y-1}_{10} + M^{y-1}_{20} + L^{y-1}_{11} + L^{y-1}_{21} + L^{y-1}_{22})2^{(m+n+2^{y-1}G)},$$

$$(OM^{y-1}_{11} + OM^{y-1}_{12})2^{(m+n+2^yG)} = (L^{y-1}_{13} + L^{y-1}_{23} + M^{y-1}_{12} + M^{y-1}_{22} + M^{y-1}_{11} + M^{y-1}_{21})2^{(m+n+2^yG)},$$

$$\begin{aligned} OL^{y-1}_{11} &= L^{y-1}_{1m,n}, \\ OL^{y-1}_{12} &= L^{y-1}_{2m,n}, \\ OM^{y-1}_{21} &= M^{y-1}_{1(m+2^{y-1}G),(n+2^{y-1}G)}, \\ OM^{y-1}_{22} &= M^{y-1}_{2(m+2^{y-1}G),(n+2^{y-1}G)}. \end{aligned} \tag{16}$$

All the symbols used in (16) represent pure binary numbers having one bit per weight.

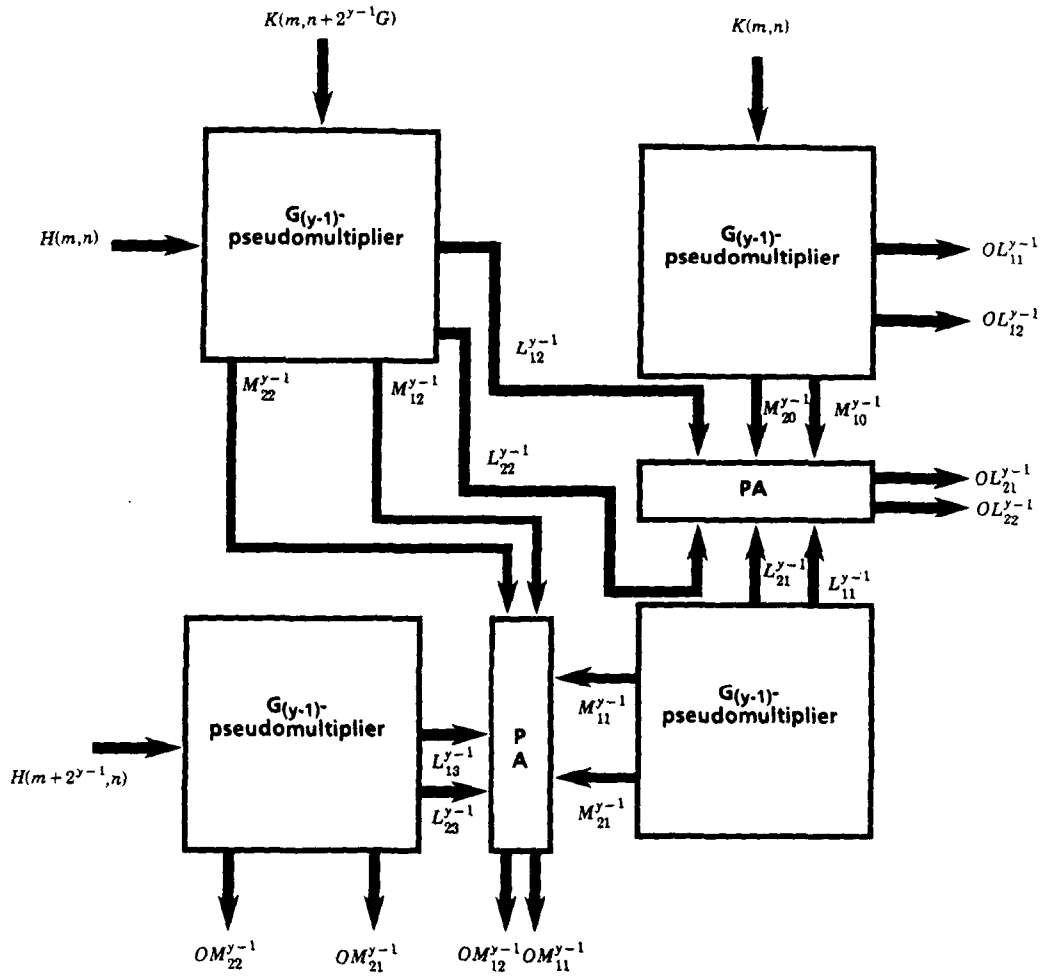


Fig. 4. G_y-pseudomultiplier.

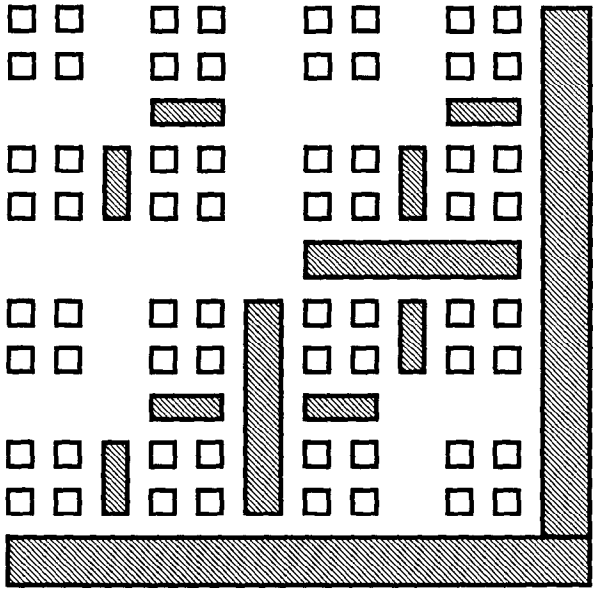


Fig. 5. Recursive multiplier layout.

The representation of the output $P^y_{m,n}$, according to the scheme proposed by **step y**, is obtained by combining the outputs of Pseudo-Adders as follows:

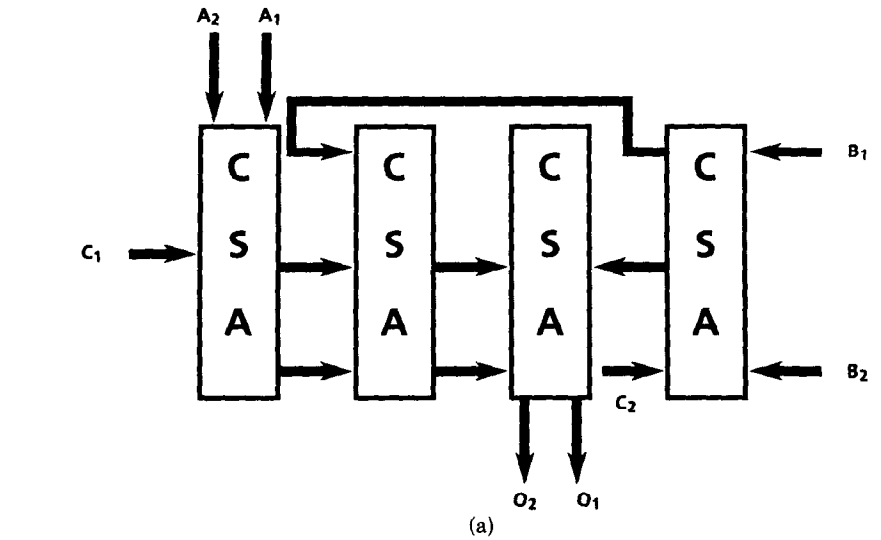
$$\begin{aligned}
2^{(m+n)}L^y_{1m,n} &= OL^{y-1}_{11}2^{(m+n)} + OL^{y-1}_{21}2^{(m+n+2^{y-1}G)},\\
2^{(m+n)}L^y_{2m,n} &= OL^{y-1}_{12}2^{(m+n)} + OL^{y-1}_{22}2^{(m+n+2^{y-1}G)},\\
2^{(m+n+2^yG)}M^y_{1m,n} &= OM^{y-1}_{11}2^{(m+n+2^yG)} + OM^{y-1}_{21}2^{(m+n+2^yG+2^{y-1}G)},\\
2^{(m+n+2^yG)}M^y_{2m,n} &= OM^{y-1}_{12}2^{(m+n+2^yG)} + OM^{y-1}_{22}2^{(m+n+2^yG+2^{y-1}G)}.
\end{aligned} \tag{17}$$

Figure 5 shows a schematic layout for a G_2-pseudomultiplier; connections between macrocells and PAs have been omitted for the sake of simplicity. The PAs are represented in Figure 5 by dashed lines. The maximum number of input addends in a PA is 6 and wires carrying pairs of addends are already ordered in such a way that most of the wires carrying bits of the same weight are adjacent.

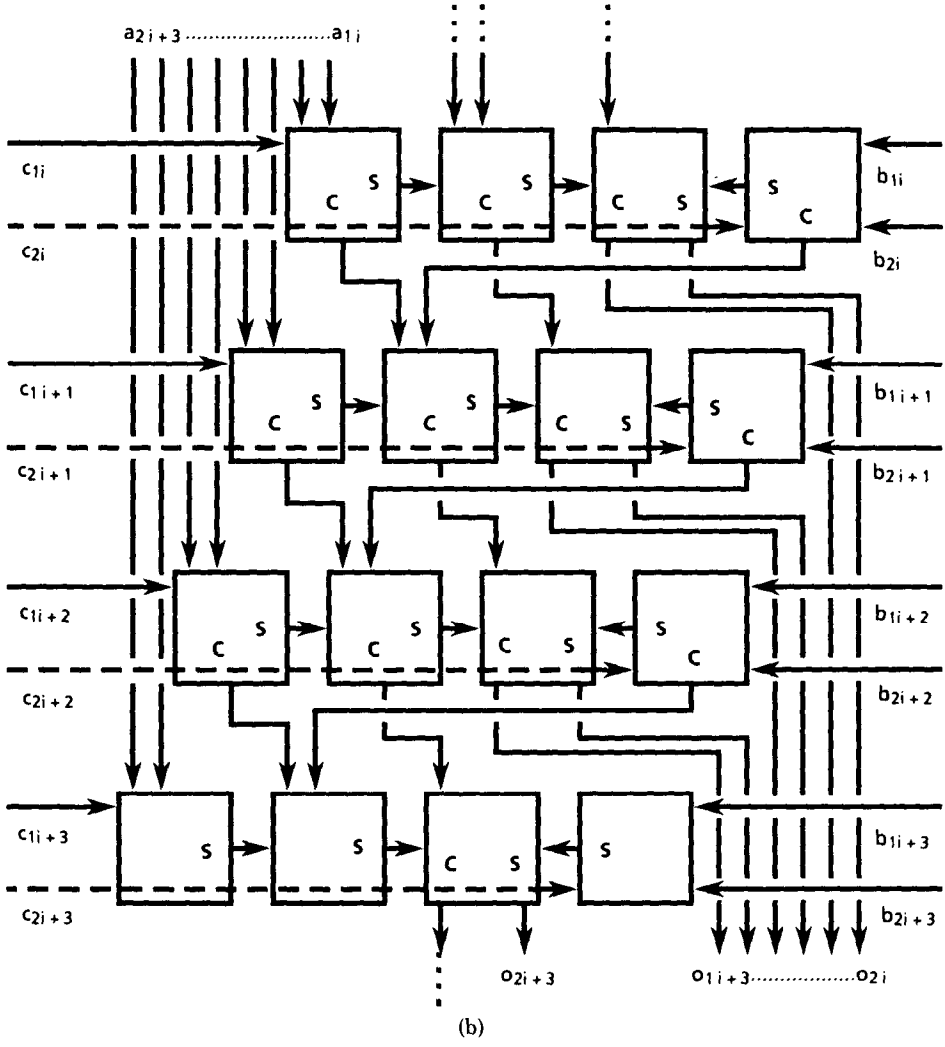
The Special-Adder (SA) layout occupies two sides of the square and is also represented by dashed lines.

The layout of a PA is shown in Figure 6. Figure 6a shows the structure of the Special Adder which uses four Carry-Save Adders for adding six numbers A_1, A_2, B_1, B_2, C_1, C_2 in order to produce two output O_1 and O_2 whose sum is equal to the sum of the six addends. Figure 6b shows the detailed implementation of the CSAs. Each cell is a full adder, the sum and carry outputs are indicated by S and C. Capital letters in Figure 6a indicate binary numbers and lower case letters in Figure 6b indicate their bits.

The time complexity of the proposed PA structure is $O(3)$. The area complexity of a PA is $O((2^{(y-1)}G)(\alpha 2^{(y-1)}G + 4)) = O(\alpha(2^{(y-1)}G)^2)$. Where α is the ratio between the area complexity of a pair of wires and the side of a G_0-pseudomultiplier divided by G.



(a)



(b)

Fig. 6. (a) Pseudoadder scheme (global scheme). (b) Pseudoadder scheme (detailed scheme).

3. TIME AND AREA COMPLEXITIES

3.1 Time Complexity

Assuming that D is the delay introduced by an AND/OR circuit implementing the basic functions of a PA and the Special Adder (SA), assuming T(G) is the delay introduced by a G_0-pseudomultiplier, if Y recursions are applied, then the multiplication delay is the sum of three contributions due to the special adder, the chain of PAs, and the G_0-pseudomultipliers. The contribution of the special adder is taken from [4]. This global delay can be expressed as follows:

$$T = (\log 2N + 3Y)D + T(G). \tag{18}$$

Logarithms are supposed to be base two unless specified otherwise.

Using the just described design approach, a cellular G_0-pseudomultiplier of G bit can be designed with a delay

$$T(G) = GD. \tag{19}$$

The G_0-pseudomultiplier could also be implemented with a Read Only Memory (ROM) making $T(G)$ independent of G. In this case the (18) can be rewritten as

$$T = (\log 2N + 3Y)D + T(\text{ROM}).$$

A technologically acceptable solution could be a ROM with a number of bits less than 2^{12}.

In order to find a relation between Y and N from (14) ones obtains

$$Y = \lceil \log(N/G) \rceil. \tag{20}$$

Where $\lceil A \rceil$ means A if A is integer or the least integer greater than A.

In order to keep the time complexity of the structure logarithmic, different conditions on G can be imposed. A very simple one is the following:

$$G = \log N. \tag{21}$$

With this assumption the global delay can be expressed as

$$T = (\log 2N + 3 \log(N/\log N) + \log N)D.$$

and the overall time complexity is $O(\log N)$. Notice that for low values of N, $3 \log(N/\log N)$ does not introduce a remarkable contribution to the overall delay.

3.2 Area Complexity

The area complexity of the recursive multiplier can be computed by inspection of Figures 5 and 6 as follows:

$$\begin{aligned} O(A) &= O(A(G_Y\text{-}pseudomultiplier) + A(Special\ Adder)) \\ &= O(4A(G_{Y-1}\text{-}pseudomultiplier)) \\ &\quad + O(N \log N) + area\ of\ 2((N/2)\ bit\ pseudoadders)) \\ &= O(16A(G_0\text{-}pseudomultiplier)) \\ &\quad + area\ of\ 2((N/2)\ bit\ pseudoadders)) \\ &\quad + area\ of\ 4((N/4)\ bit\ pseudoadders)) + O(N \log N). \end{aligned}$$

In general, there are $(N/G)^2$ G_0-pseudomultipliers whose area complexity is $O(N^2)$.

There are $Y \approx \log(N/\log N)$ levels of pseudoadders. At level Y there are two pseudoadders. Each one of them has an area complexity that can be computed from the scheme of Figure 6. The horizontal size is proportional to $2\alpha N/2$ and the vertical size is proportional to $N/2$. Thus the area complexity at level Y is:

$$O\left(2 * 2a * \frac{N^2}{2^2}\right) = O\left(a\frac{N^2}{2}\right).$$

Given the proposed structure and the cell design proposed in [8], α is the ratio between the area complexity of a wire and the area complexity of a size of a cell of the G_0-pseudomultiplier.

At level $Y - 1$ there are 2^2 PAs handling $N/2^2$ wires. The area complexity at level $Y - 1$ is

$$O\left(2 * 2a * \frac{N^2}{2^{2*2}}\right) = O\left(a\frac{N^2}{2^2}\right).$$

The overall area complexity of the PAs is

$$A_{PA} = O\left(aN^2\left(\sum_{i=1}^{Y} \frac{1}{2^i}\right)\right) = O(aN^2). \tag{22}$$

Equation (22) does not take into account the space that remains free in Figure 5. Part of this space is occupied by wires that connect pseudomultipliers with pseudoadders.

Taking this space into account in the evaluation of area complexity represents a worst-case situation in which no attempt is made for squeezing G_0-pseudomultipliers and pseudoadders in order to obtain an optimal layout with $O(N^2)$ area complexity. In the worst case situation, corresponding to the layout of Figure 5, it is important to notice that each square containing four G_y-pseudomultipliers has two strips whose sizes are

$$\frac{N}{2^{Y-y}} \quad \text{and} \quad 2a\frac{N}{2^{Y-y+1}}.$$

Thus, the overall area complexity can be computed as follows:

$$\begin{aligned} A_{PA} &= 2 * 2a * N^2\left(\frac{1}{2^0 * 2^1} + 2^2\frac{1}{2^1 * 2^2} + 2^4\frac{1}{2^2 * 2^3} + \cdots\right) \\ &= 2aN^2(1 + 1 + \cdots) \\ &= 2aN^2\log R. \end{aligned}$$

In the worst case, the area complexity of the G_Y-pseudomultiplier is

$$A = O(N^2(1 + 2aY)). \tag{23}$$

The area complexity of the proposed structure can be assumed to be $O(N2)$ as far as

$$\log(N/\log N) \leq (1/2a). \tag{24}$$

Following the cell design proposed in [8], each cell of a G_0-pseudomultiplier contains four 16-input multiplexers and circuits implementing functions of four variables. Based on the above considerations it can be assumed that $\alpha \approx 0.025$ which makes (24) an acceptable condition for a large class of practical multipliers.

4. MULTIPLICATION OF TWO'S COMPLEMENT NUMBERS

From Figure 2a it appears that a G_0-pseudomultiplier can accept two additive inputs, a G bit number along the vertical inputs lines and a $2G$-bits number along the diagonal lines. In the layout sketched in Figure 5 each square represents a G_0-pseudomultiplier each one of which can accept two additive inputs. In particular, the lowermost row and the leftmost column of G_0-pseudomultipliers can accept two addends whose bit weights range from 2^{N-1} to 2^{2N-2}. We will show how these inputs can be used for multiplying two binary numbers with negative numbers represented in the two's complement notation using the Baugh and Wooley [1] algorithm. For the sake of clarity the algorithm will be summarized in the following.

Let now assume that H and K are N bits two's complement numbers:

$$\begin{aligned} H &= -h_{N-1}2^{N-1} + \sum_{i=0}^{N-2} h_i 2^i, \\ K &= -k_{N-1}2^{N-1} + \sum_{i=0}^{N-2} k_i 2^i. \end{aligned} \tag{25}$$

The product P can then be expressed as follows:

$$\begin{aligned} P &= P_1 + P_2 + P_3 + P_4 + P_5 + P_6, \\ P_1 &= 2^{2N-1}, \\ P_2 &= (\overline{h_{N-1}} + \overline{k_{N-1}} + h_{N-1}k_{N-1})2^{2N-2}, \\ P_3 &= \sum_{i=0}^{N-2}\sum_{j=0}^{N-2} h_i k_j 2^{i+j}, \\ P_4 &= \sum_{j=0}^{N-2} h_{N-1}\overline{h_j}2^{N-1+j}, \\ P_5 &= \sum_{i=0}^{N-2} k_{N-1}\overline{k_i}2^{N-1+i}, \\ P_6 &= (h_{N-1} + k_{N-1})2^{N-1}. \end{aligned} \tag{26}$$

The term P_3 is the main product performed by the G_Y-pseudomultiplier. The term P_4 can be added in the leftmost column of cells of G_0-pseudomultipliers by introducing an array of cells made by an inverter and a two input AND gate fed by h_{N-1} and the complement of h_j. In the same way, the term P_5 can be added in the lowermost row.

The terms P_1, P_2 and P_6 can be created by some small additional hardware and added to SA.

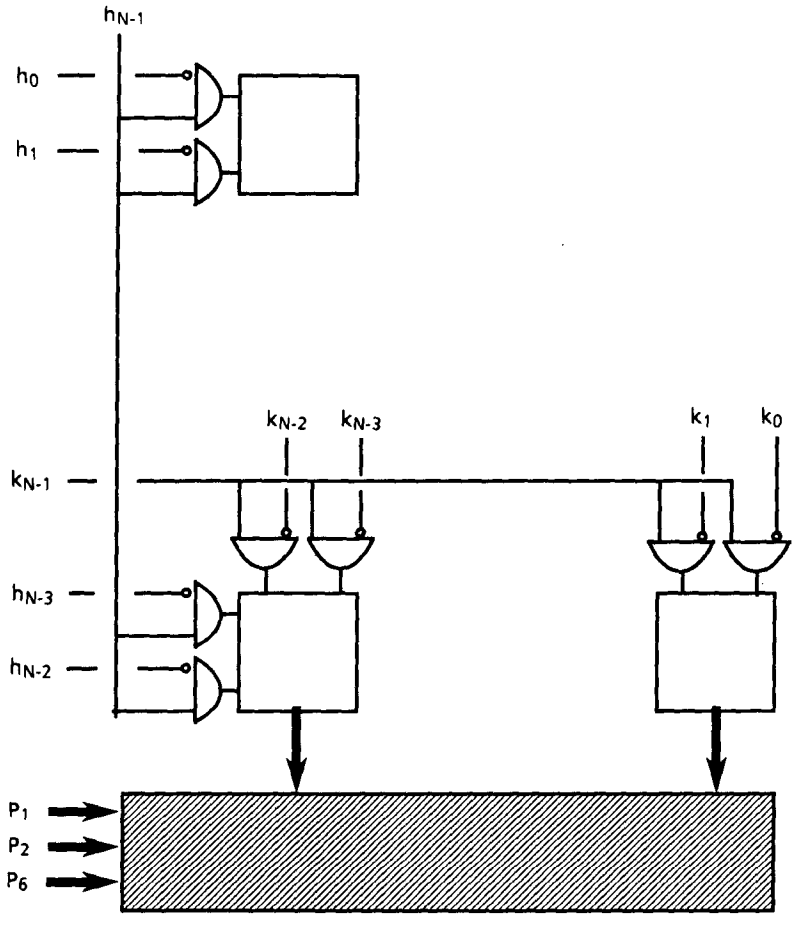


Fig. 7. Scheme for two's complement multiplication.

Figure 7 shows the details of the modification introduced in the leftmost column and the lowermost line of G_0-pseudomultipliers in order to make the structure capable of multiplying two's complement numbers. By inspection of Figure 7 it is clear that the introduced modifications do not affect the delay nor the area complexities of the multiplier as evaluated in Section 3.

5. PIPELINING

Let PE be the period of a multiplier, defined as the time between completion of successive multiplication instances. If the number of bits of the multiplier and the application for which the multiplier is designed are such that the combinational array introduced so far can complete an operation before a new operation is started, then registers can be introduced only for storing the factors and the product. In this case, the period complexity of the array would be

$$\mathrm{PE} = 1. \tag{27}$$

If the application requires a period between the output of two successive results to be less than the multiplication time, then the array can be *pipelined* for reducing the idle time of the circuit cells.

Concepts derived from retiming transformation [14] can be applied to the recursive multiplier in order to implement pipelining. Following a recent suggestion by Hawck et al. [12], pipelining can be implemented by first adding registers on the inputs and then retiming to minimize the period.

The *degree of pipelining* can be defined as the maximum number of cells between any pair of registers. For our specific application, there are two types of cells, normally, the cells of the G_0-pseudomultiplier and the carry-save adders used in the Pseudo-Adders and in the Special Adder.

The most effective level of pipelining depends on many practical and technological considerations. A practically acceptable degree of pipelining given the actually available adders and multiplexers is four. This implies that registers can be placed at the output of the pseudoadders whose structure is proposed in Figure 6 but there is no need of providing registers inside them.

Figure 8 shows a rearrangement of the layout shown in Figure 5 where G_0-pseudomultipliers are represented by squares, pseudoadders are represented by rectangles and the wires carrying the bits of each of the binary numbers representing a PA's output are represented by a single arrow. A black arrow represents also an array of registers on the corresponding wires.

As far as $G < 4$, there is no need of introducing registers inside the G_0-pseudomultipliers. It suffices to introduce registers at their outputs. From inspection of Figure 8 one can conclude that extra registers are required at the output of the PA feeding the SA in order to synchronize the appearance of the bits of the same product at the input of SA. The number of extra registers required is zero for the rightmost and the lowermost SA and increases by one going leftward and upward. These registers are not shown in Figure 8 for the sake of simplicity. Pipelining SA has been discussed elsewhere [2] and won't be discussed here.

Should the G_0-pseudomultiplier be pipelined, the same technique proposed by Hawck et al. [12] can be used for placing the registers. In any case, a period complexity $\mathrm{PE} = O(1)$ can be achieved.

Several VLSI Figures of Merit have been proposed. The ones that have been mostly discussed are:

$$\begin{aligned} \mathrm{FM}_a &= AT^2(\mathrm{PE})^2, \\ \mathrm{FM}_b &= A(\mathrm{PE})^2, \\ \mathrm{FM}_c &= A(\mathrm{PE})T. \end{aligned} \tag{28}$$

Lower bounds for binary multipliers have been derived for each VLSI figure.

The recursive structure proposed meets these lower bounds with the complexities:

$$\begin{aligned} \mathrm{A} &= N^2, \\ \mathrm{T} &= \log N, \\ \mathrm{PE} &= 1. \end{aligned} \tag{29}$$

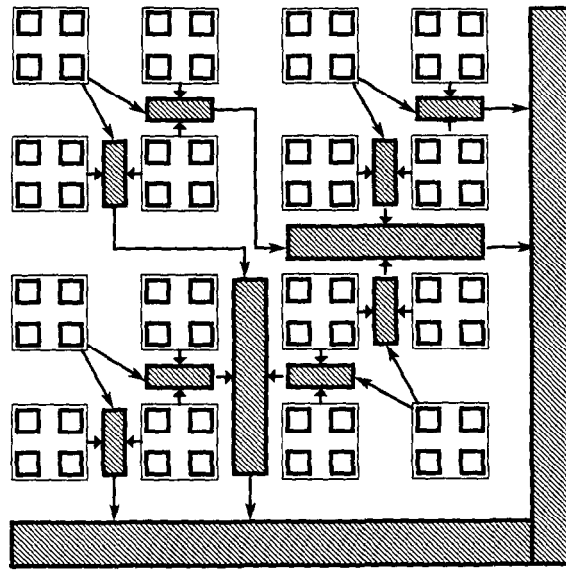


Fig. 8. Pipelined multiplier layout.

6. SOME APPLICATIONS OF THE RECURSIVE STRUCTURE AND CONCLUSIONS

Figure 9 shows a layout for a structure based on G_Y-pseudomultipliers capable of performing a single double-precision multiplication or four single-precision multiplications. Binary numbers are represented with a single arrow.

S_1, S_2, S_3, S_4 represent single precision outputs, DP represents a double precision output. Essentially, four SAs are included for single precision with a global contribution to the whole area complexity of $A_{4SA} = 4N \log N$. These single-precision SAs are bypassed when double-precision multiplication has to be performed. External switches must be added to send the factor bits to the four G_Y-pseudomultipliers and to select the outputs to be stored. The complexity of these switches, which are now shown in Figure 9 for the sake of clarity, is $O(N)$. Thus, the overall area complexity remains $O(N^2)$ and the time complexity remains unchanged.

Convolution and digital filter circuits can be conceived with the same approach. G_Y-pseudomultipliers can be used for producing two numbers whose sum is equal to the product. Partial additions of products can be performed using pseudoadders. A single Special Adder can be used for producing the final output as in [10].

Using the same layout shown in Figure 5, a C-order convolution can be performed with a latency complexity $\log CN$ and an area complexity CN^2. A similar approach can be taken for digital filter design.

In conclusion, the recursive design approach introduces some new ideas that can be useful for an automatic layout design. Another potentially useful idea presented is that of combining multiplexer-based macrocells with pseudoadders. Macrocells based on multiplexers allow a fast propagation of changes of variables affected by a large delay, by applying them to the address select inputs while

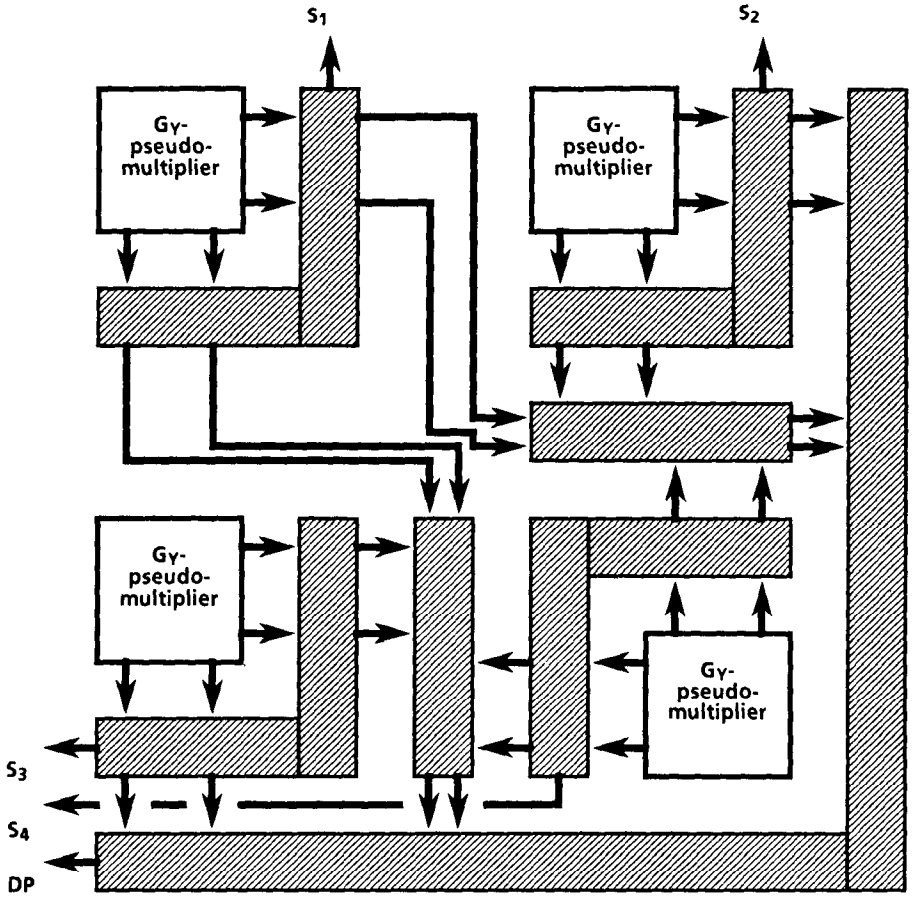


Fig. 9. Single and double precision operation.

using simpler and slower circuits applied at the multiplexer inputs for handling factor bits. Paths containing a few macrocells in cascade can be tolerated as far as their delay is not the predominant contribution to the overall multiplier delay. Efficient pseudoadders with constant delay can be used for adding up the outputs of chains of macrocells in order to produce two numbers whose sum is equal to the desired product.

A recursive procedure can systematically place pseudoadders in a regular layout of macrocells making the maximum length of a chain of pseudoadders proportional to log N.

It is hoped that these ideas will suggest improvements on the layout design of circuits other than multipliers.

REFERENCES

1. BAUGH, C. R. AND WOOLEY, B. A. A two's complement parallel array multiplication algorithm. *IEEE Trans. Comp. C-22*, (Dec. 1973), 1045–1047.
2. BRENT, R. P. AND KUNG, H. T. The chip complexity of binary arithmetic. In *Proceedings of*

the 12th Annual ACM Symposium on Theory of Computing (Los Angeles, Cal., April 28–30), ACM, New York, 1980, pp. 190–200.

3. BURTON, D. P., BYRNE, P. C., AND NOAKS, D. R. A multiplier for complex binary numbers. *Electron. Eng.* (Apr. 1970), 71–73.
4. CAPELLO, P. R. AND STEIGLITZ, K. A VLSI layout for a pipelined Dadda multiplier. *ACM Trans. Comp. Syst. 1*, (May 1983).
5. DADDA, L. Some schemes for parallel multipliers. In *Computer Design Development*, E. E. Swartzlander, Jr., Ed. Hayden Book, Rochelle Park, N.J., 1976.
6. DEAN, K. J. Cellular multiplier subarrays: a critical-path approach to propagation time. *Electron. Lett. 7*, (Feb. 1971), 75–77.
7. DE MORI, R. Suggestion for an IC fast parallel multiplier. *Electron. Lett. 5*, Jan. 1969, 50–51.
8. DE MORI, R. AND CARDIN, R. A parallel multiplier based on multiplexers. *Signal Process. 6*, June 1984, 213–223.
9. DE MORI, R. AND SERRA, A. A parallel structure for signed-number multiplication and addition. *IEEE Trans. Comp. C-21*, (Dec. 1972), 1453–1454.
10. DE MORI, R., RIVIORA, S., AND SERRA, A. A special-purpose computer for digital processing. *IEEE Trans. Comp. C-24*, (Dec. 1975), 1202–1211.
11. DE MORI, R. AND VENKATESH, K. On the testability of macro cellular multipliers. Internal Rep., Concordia University, May 1985.
12. HAWCK, C. E., BANIJI, C. S., AND ALLEN, J. The systematic exploration of pipelined array multiplier performance. In *Proceedings of the IEEE International Conference on Acoustic Speech and Signal Processing ICASSP-85* (Tampa, Fla., 1985) pp. 1461–1464.
13. HWANG, H. *Computer arithmetic: Principles, architecture and design.* John Wiley, New York, 1979.
14. LEISERSON, C., ROSE, F., AND SAXE, J. Optimizing synchronous circuitry by retiming. In *Third Caltech Conference on VLSI*, California Institute of Technology (Pasadena, Calif., March 1983).
15. LING, H. High speed computer multiplication using a multiple-bit decoding algorithm. *IEEE Trans. Comp. C-19*, (Aug. 1970), 706–709.
16. LUK, W. K. A regular layout for parallel multiplier of $O(\log^2 N)$ time. In *VLSI Systems and Computations*, H. T. Kung, R. F. Sproull, and G. L. Steele, Jr., Eds. Computer Science Press, Inc. Carnegie-Mellon Univ., 1981.
17. MAJITHIA, J. C. AND KITAI, R. An iterative array for multiplication of signed binary numbers. *IEEE Trans. Comp. C-20*, (Feb. 1971), 214–216.
18. PREPARATA, F. P. A mesh-connected area-time optimal VLSI multiplier of large integers. *IEEE Trans. Comp. C-32*, (Feb. 1983), 194–198.
19. SINGH, S. AND WAXMAN, R. Multiple operand addition and multiplication. *IEEE Trans. Comp. C-22*, (Feb. 1973), 113–120.
20. STENZEL, W. J., KUBITZ, W. J., AND GARCIA, G. H. A compact high-speed parallel multiplication scheme. *IEEE Trans. Comp. C-26*, (Oct. 1977), 948–957.
21. WALLACE, C. S. A suggestion for a fast multiplier. *IEEE Trans. Electron. Comp.* (Feb. 1964), 14–17.

Received May 1983; revised May 1985; accepted August 1985