A Compact High-Speed Parallel Multiplication Scheme

WILLIAM J. STENZEL, WILLIAM J. KUBITZ, MEMBER, IEEE, AND GILLES H. GARCIA

Introduction

SCHEMES for parallel multiplication are roughly divisible into two classes—iterative arrays of cells [Fairchild 9344 and Advanced Micro Devices 25S05], [1]-[4], and generation of a matrix of partial-product terms with subsequent reduction of the matrix by means of pseudoadders [5]–[10]. Array schemes are attractive for their compactness and use of one basic circuit type, but their speed of operation increases linearly with operand length and is slow for large words [6]. Matrix generation-reduction schemes are much faster for larger operands since their speed of operation increases with the log of the operand length [11]. Traditional forms of the matrix generation-reduction scheme employ AND gates as 1-by-1-bit multipliers in forming the partial-product matrix and use full adders to reduce the matrix; this form is not nearly as compact as the array scheme. Integrated circuits now exist which can form partial products for operands larger than 1 bit [Texas Instruments 74S274] or reduce larger portions of the matrices than is possible with full adders [12], [Texas Instruments 74S275]. This allows the fabrication of generation-reduction-type multipliers which rival array-type multipliers in compactness.

This paper deals with a compact form of generation-reduction-type multiplier. The consequences of employing larger partial-product generation circuits and more general reduction circuits are considered; possible implementations of these circuits are discussed. An algorithm for the design of multipliers using these circuits is presented and is used to obtain measures of merit for several multiplication schemes. Finally, the fabrication of a prototype 24 × 24-bit multiplier employing such circuits is described and its performance is evaluated. All schemes described are for unsigned (i.e., sign-magnitude) multiplication. If 2's complement multiplication is desired, the algorithms presented in [11] and [13] may be used to generate additional partial-product terms that will produce a 2's complement result when either added to the unsigned product or incorporated into the matrix and reduced along with the rest of the terms.

General Considerations

Fig. 1 shows diagrammatically the process of multiplying two 6-bit unsigned numbers. The dot representation used in Fig. 1 is a convenient means of depicting the multiplication process and will be used throughout this discussion. The product terms shown are most often generated by means of AND gates (1 × 1 multipliers) and, using the approach of Wallace and Dadda [9], [5], full adders are used to reduce the partial-product matrix by summing the columns. The complete reduction is shown in Fig. 2 for this example. A carry look-ahead adder is normally used to reduce the final two rows. Note that for the 1 × 1 multiplier case the matrix height is given by

$$\begin{aligned} h_i &= i + 1, && \text{for } i = 0, \cdots, n-1 \\ h_i &= 2n - 1 - i, && \text{for } i = n, \cdots, 2n-2 \end{aligned}$$

where h_i is the height of the column of weight 2^i and n is the word length.

This scheme may be improved by using $m \times m$ rather than 1 × 1 multipliers for the product-term generation. The 256 × 8 read-only memories (ROM's) may be programmed with the multiplication tables for pairs of 4-bit operands. The two 4-bit operands are input as an 8-bit address and their 8-bit product is the output. These ROM's may be used as 4 × 4 multipliers to generate partial products. The height of the columns of the matrix for $m \geq 2$ is given by

$$\begin{aligned} h_i &= 2\left\lfloor \frac{i}{m} \right\rfloor + 1, && \text{for } i = 0, \cdots, n-1 \\ h_i &= 2\left\lfloor \frac{(2n-1-i)}{m} \right\rfloor + 1, && \text{for } i = n, \cdots, 2n-1 \end{aligned}$$

where h_i and n are the same as before [14].[1]

The savings obtained for generation lead one to seek similar savings for reduction. Accordingly, we note that 1K × 4-bit ROM's may be programmed to treat the 10 address lines as two adjacent columns of 5 bits each and perform a table lookup on the sum. A ROM so programmed can be used to reduce a matrix in a fashion similar to that of full adders. Note that the maximum sum is 15, which requires exactly 4 bits for its binary representation. This provides

[1] $\lfloor x \rfloor$ is the largest integer less than or equal to x; the "floor" of x as in APL.

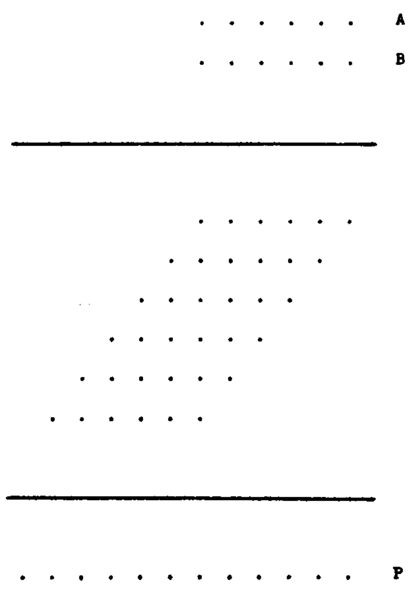

Fig. 1. Diagramatic representation of 6 × 6 multiplication.

complete utilization of the 10-bit address field and the 4-bit output field. This tool may be applied to a 12 × 12-bit multiplication with the result shown in Fig. 3. The first level in the figure corresponds to the full matrix resulting from the use of 4 × 4 multiplier modules. The full n^2 expansion would be 12 bits high and contain twice as many terms. The second row shows the two-row matrix resulting from the use of ROM's for reduction.

Generalized Counters

Dadda [5] has introduced the notion of a (c,d) counter as a combinatorial network which receives c bits of equal weight as input and produces a d-bit word corresponding to their sum as output. A full adder, for example, would be termed a (3,2) counter. The value of the output is

$$v = \sum_{i=0}^{c-1} b_i$$

where b_i denotes the binary value of bit i of the input column and v denotes the value of the d-bit output. The number of output bits must be sufficient to represent all possible sums of c bits, and hence

$$2^d - 1 \geq c.$$

This class of counters may be extended to include counters which receive several successively weighted input columns and produce their sum, taking the weighting into account. We denote counters of this type as

$$(c_{k-1}, c_{k-2}, \cdots, c_0, d)$$

counters, where k is the number of input columns, c_i is the number of input bits in the column of weight 2^i, and d is the number of bits in the output word. The value of the output is

$$v = \sum_{i=0}^{k-1} \sum_{j=0}^{c_i-1} b_{ij} 2^i$$

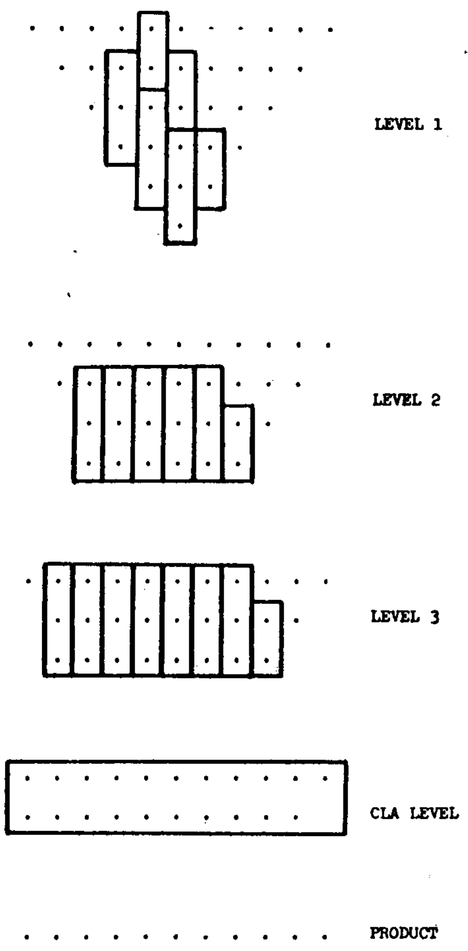

Fig. 2. Wallace tree reduction for 6 × 6 multiplication. Boxes indicate adders.

where b_{ij} denotes the value of bit j in column i. Again d is subject to the constraint that

$$2^d - 1 \geq \sum_{i=0}^{k-1} c_i 2^i.$$

Examples of several counters are shown in Fig. 4. The enclosed dots represent the configuration of the input bits, and are followed by the resultant output bits. The counters shown (with the exception of the (3,3,3,3,6)) all have complete utilization. Complete utilization is not necessary but it is desirable. There are many underutilized arrangements and, although useful in certain applications, they will not be discussed here.

The effect of a series of counters acting on adjacent sets of input columns is shown in Fig. 5. The inputs to the counters are shown first, followed by an equivalent representation of the output. Let us refer to the number of resulting output rows as s. We see, then, that a series of (7,3) counters can reduce a matrix 7 rows high to a matrix 3 rows high ($s = 3$), or a series of (5,5,4) counters can reduce 5 rows to 2 rows ($s = 2$). A string of (2,2,2,3,5) counters can reduce 3 rows to 2 rows by virtue of the fact that an extra input bit is consumed in the low order position, compensating for the carry out of the previous counter.

Let us further restrict ourselves to counters with input

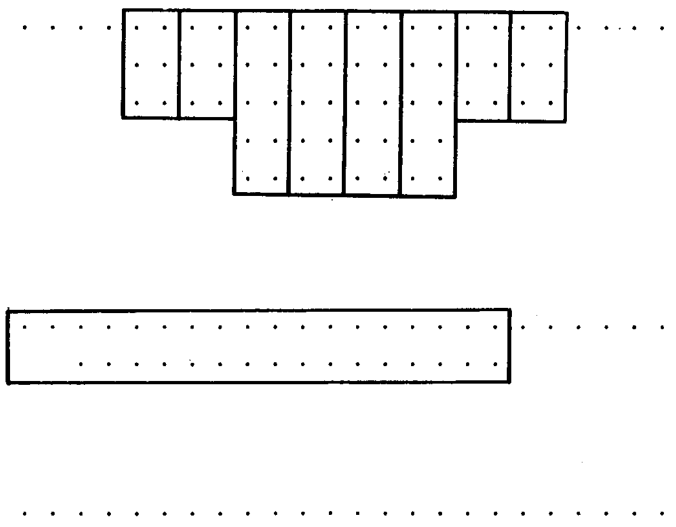

Fig. 3. 12 × 12-bit partial product reduction using (5,5,4) counters. Partial products are generated by 4 × 4 multipliers.

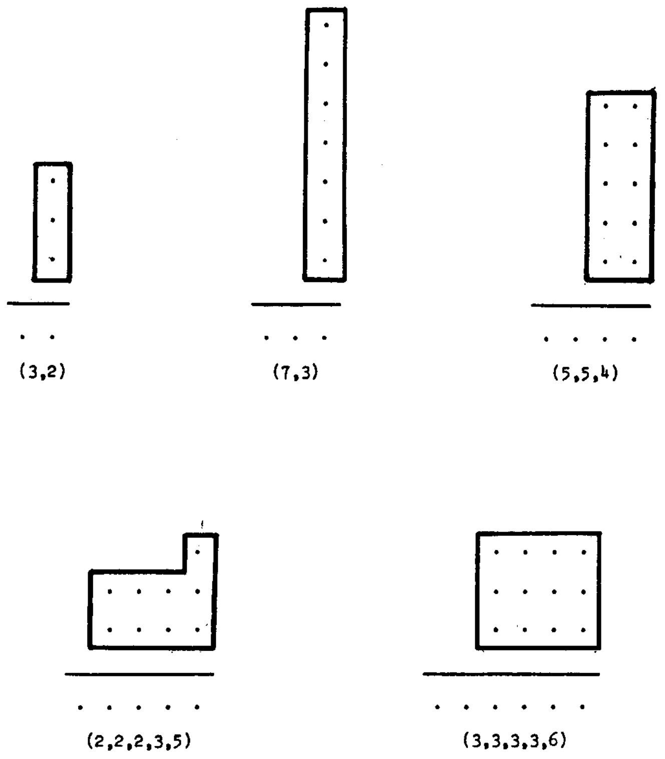

Fig. 4. Some generalized counters.

columns of equal height. Equal column counters are not the only ones of interest, but are a convenient tool for reducing the regular portions of a matrix and are thus more useful. The regularity of these counters permits us to make the following observations, which are not necessarily true of unequal column counters.

Equal column counters consume a rectangular matrix segment of k columns by r rows, where

$$r = c_{k-1} = c_{k-2} = \cdots = c_0.$$

As Fig. 5 shows, a string of counters produces d-bit outputs at intervals of k bits. The outputs align themselves such that no more than $\lceil d/k \rceil$ outputs contribute to a given column.[2] Hence, the number of rows, s, of the output matrix is simply

$$s = \left\lceil \frac{d}{k} \right\rceil.$$

The height of the resulting output matrix is determined by the number of input and output columns, subject to the constraint that the sum of the r input rows is representable

[2] $\lceil x \rceil$ is the smallest integer greater than or equal to x; the "ceiling" as in APL.

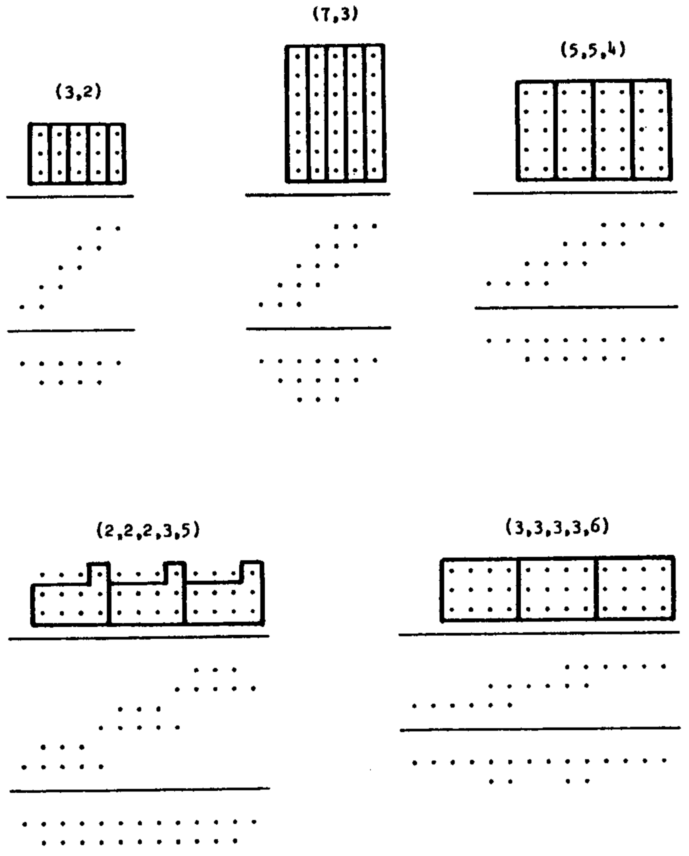

Fig. 5. Effect of a series of adjacent counters.

in d bits. The number of resultant output rows has a direct bearing on the number of stages of counters needed to reduce a large matrix, so it is desirable that the number of output rows be small. If we let

$$v_r = 2^k - 1$$

denote the maximum possible value of a single input row and

$$v_o = 2^d - 1$$

denote the maximum representable output value, the constraint on the number of input rows may be expressed as

$$v_o \geq r v_r$$

or

$$r \leq \frac{v_o}{v_r} = \frac{2^d - 1}{2^k - 1}.$$

If d is not divisible by k, the output matrix will be somewhat sparse (e.g., the output of a string of (3,3,3,3,6) counters consists of alternate 2-bit segments of height 1 and height 2). This may be used to advantage in certain situations, such as the reduction of a 6 row matrix by means of (3,3,3,3,6) counters [14]. Normally, it is advantageous to produce a matrix of uniform height since it can be reduced by a single counter type.

These considerations lead us to the notion of a maximally efficient counter as one which produces a uniform output matrix while consuming the largest possible regular portion of the input matrix; i.e., it should have equal columns with the largest possible number of rows. Disregarding cases where strings of counters may be stacked and skewed, as in the case previously mentioned, this means that the number of output columns should be a multiple of the number of input columns, or

$$d = sk.$$

The counter will consume the largest possible portion of the matrix when

$$v_o = r v_r$$

or

$$r = \frac{v_o}{v_r} = \frac{2^d - 1}{2^k - 1} = \frac{2^{sk} - 1}{2^k - 1} = 2^{(s-1)k} + 2^{(s-2)k} + \cdots + 1.$$

Maximally efficient equal column counters are given in Table I for the first few values of k and s. The only currently implementable ones are the (3,2), the (7,3), the (5,5,4), and possibly the (15,4) counters. The number of input rows increases rapidly with both the number of input columns and output rows, hence any implementation of counters with a larger k or s than those previously mentioned would probably be less than maximally efficient.

Number of Levels for Reduction

A matrix r bits high can be reduced to one s bits high through the use of one level of counters. We now consider the number of levels of counters required to reduce a ma-

TABLE I
Maximally Efficient Counters with Equal Columns

	INPUT		OUTPUT		REQUIRED ROM SIZE
	Columns	Rows	Columns	Rows	
	k	r	d	s	Words x Bits
*	1	3	2	2	8 x 2
*	2	5	4	2	1024 x 4
	3	9	6	2	2^{27} x 6
	4	17	8	2	2^{68} x 8
	5	33	10	2	2^{165} x 10
*	1	7	3	3	128 x 3
	2	21	6	3	2^{42} x 6
	3	73	9	3	2^{219} x 9
	4	273	12	3	2^{1092} x 12
*	1	15	4	4	2^{15} x 4
	2	85	8	4	2^{170} x 8
	3	585	12	4	2^{1755} x 12
	1	31	5	5	2^{31} x 5
	2	341	10	5	2^{682} x 10

* Denotes currently realizable counters.

trix more than r bits in height to one of s bits. Denote the maximum height of the matrix that may be reduced to s bits using j levels of counters as l_j. Note that $l_0 = s$ and $l_1 = r$. Knowing l_j we may determine l_{j+1} by observing that the l_j bits represents the output of a stack of $\lfloor l_j/s \rfloor$ counters at level $j + 1$ plus a residue of (l_j) mod s bits which were not reduced by counters (Fig. 6). The $\lfloor l_j/s \rfloor$ counters each consume r bits, so

$$l_{j+1} = r \left\lfloor \frac{l_j}{s} \right\rfloor + (l_j) \bmod s.$$

For a (5,5,4) counter the maximum reduction height sequence is

$$2,5,11,26,65,\cdots$$

and for a (7,3) counter it is

$$3,7,15,35,79,\cdots.$$

A rough approximation of the first few terms of the sequence is

$$s, s\left(\frac{r}{s}\right), s\left(\frac{r}{s}\right)^2, s\left(\frac{r}{s}\right)^3, \cdots.$$

Hence, the number of levels of counters required to reduce a matrix h bits high to s rows is roughly $\log_{(r/s)}(h)$ levels.

Not all counters give a reduction to 2 rows. The (7,3) counters give a reduction to only 3 rows and the final reduction to 2 rows must be accomplished by means of an additional stage of counters. Other sequences may be obtained by combining two or more types of counters. For example, if the final level of a series of (5,5,4) counters is followed by an additional level of (3,2) counters, the final reduction height and hence the first term of the sequence for the (5,5,4) counters becomes 3 rather than 2. The sequence for the (3,2) level is 2, 3 and for the (5,5,4) levels is 3, 6, 15, 36, 90, $\cdots$, giving an overall sequence of 2, 3, 6, 15, 36, $\cdots$. Thus, if we wish to employ (5,5,4) counters for the reduction of a matrix that is initially 15 bits high to 2 rows, we may use either 3 levels of (5,5,4) counters or 2 levels of (5,5,4) counters followed by one level of (3,2) counters. Assuming that (3,2) counters have a shorter propagation delay than (5,5,4) counters, the second approach will be slightly faster than the first. Hence, we can tailor a sequence to fit the requirements of a particular matrix by mixing various counter types.

Reduction Algorithm

The heuristic rule for the reduction of a matrix with (3,2) counters as presented by Dadda [5] is directly applicable to reduction by generalized counters. Dadda's rule was to find the largest term l_j in the counter's sequence which is less than the original matrix height and to reduce the matrix only to height l_j. Each successive level of counters then reduces the matrix to the height given by the next smaller term in the series; i.e., $l_j, l_{j-1}, \cdots, l_0$. In the more general situation the sequence may be characteristic of more than one type of counter, as in the example of the previous section.

The actual placement of the counters at a given reduction level is determined by traversing the matrix from right to left, inserting counters as needed to reduce columns to the targeted reduction height. The entire process may be stated algorithmically as follows.

Denote the desired word length as n, the multiplier module size as m, and the counters at a given level as $(c_{k-1}, \cdots, c_0, d)$ as before, where k is the number of input columns, d is the number of output columns, and $s = \lceil d/k \rceil$ is the number of output rows. Further, let h_i for $i = 0, \cdots, 2n - 1$ denote the current height of column i of the matrix and let the

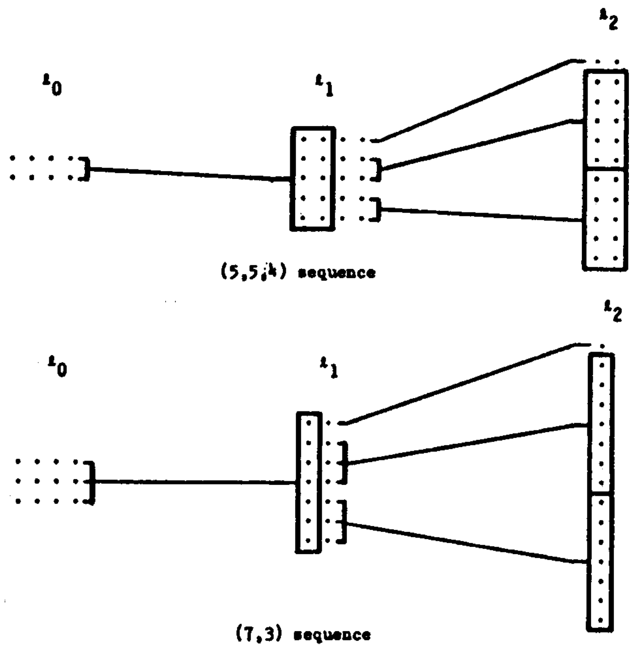

Fig. 6. Examples of multilevel reduction by (5,5,4) and (7,3) counters.

variables H and T denote the matrix height before and after the reduction, respectively.

1) Initialize h_i and H as follows.

a) If $m = 1$

$$h_i = i + 1, \qquad \text{for } i = 0, \cdots, n-1$$
$$h_i = 2n - 1 - i, \qquad \text{for } i = n, \cdots, 2n-2$$
$$H = n.$$

b) If $m \geq 2$

$$h_i = 2\left\lfloor \frac{i}{m} \right\rfloor + 1, \qquad \text{for } i = 0, \cdots, n-1$$

$$h_i = 2\left\lfloor \frac{(2n-1-i)}{m} \right\rfloor + 1, \qquad \text{for } i = n, \cdots, 2n-1$$

$$H = \frac{2n}{m} - 1.$$

2) Set T to the largest term in the reduction series which is less than H; i.e.,

$$T = l_j, \qquad \text{for } j \text{ such that } l_j < H \text{ and } l_{j+1} \geq H.$$

3) If $H = 2$, then terminate the algorithm; otherwise perform step 3a) for $i = 0, \cdots, 2n-1$.

a) If $h_i \leq T$, do nothing; otherwise insert a counter at this point, adjust the column heights accordingly, and repeat step 3a) for the new height of the same column i. The rules for adjusting column heights are:

$$h_{i+j} = \max\,[T, (h_{i+j} - c_j + 1)], \qquad \text{for } j = 0, \cdots, k-1$$
$$h_{i+j} = h_{i+j} + 1, \qquad \text{for } j = k, \cdots, d-1.$$

4) Set H to T and T to the next smaller term in the reduction series. This term is given by

$$T = s\left\lfloor \frac{T}{r} \right\rfloor + (T) \bmod r$$

where the s and r values characterize the counters that will be used at the next level.

5) Go to step 3.

This process is easily implemented on a computer.

An example of a reduction is shown in Fig. 7 for a 32 × 32-bit multiplication using 4 × 4-bit multipliers and (5,5,4) counters. The height of the matrix is initially 15 and the reduction sequence is 2, 5, 11, 26. In some cases, a (5,5,4) counter has been underutilized so that the reduction proceeds only to the desired height.

Previously, it was mentioned that a reduction from 15 rows to 2 rows could be accomplished by using 2 levels of (5,5,4) counters and one level of (3,2) counters. This is shown in Fig. 8. The sequence is 2, 3, 6, 15. If there is a need to economize on the number of large counters, a slight improvement can be obtained by using smaller counters to replace underutilized large counters wherever possible.

Comparison of Several Schemes

The algorithm for partitioning multipliers has been used to generate partitions for multipliers using various combinations of counters and multiplier modules. The counters and multiplier modules considered are all currently available. Hence we may use the characteristics of the individual components (TTL in this case) in conjunction with the component counts produced by the respective multiplier generating programs to obtain realistic measures of speed, power consumption, relative circuit board area, and cost.

Four schemes are compared here.

1) Generate partial products with 4 × 4 multiply modules and reduce with both (5,5,4) and (3,2) counters, using the (3,2) counters wherever possible.

2) Generate matrix with 4 × 4 multiply modules and reduce with both (7,3) and (3,2) counters, using the (3,2) counters wherever possible.

3) Generate matrix with 4 × 4 multiply modules and reduce with (3,2) counters exclusively.

4) Generate matrix with 1 × 1 multiply modules and reduce with (3,2) counters exclusively.

The components involved are implemented in several versions of TTL, so for purposes of normalization their propagation delays are characterized in terms of equivalent logic levels, with each logic level having a standard gate delay of 10 ns. An attempt has also been made to normalize the power consumption to that characteristic of a standard TTL implementation of the IC's. It is assumed that the circuits are packaged in standard dual in-line packages (DIP's) and that several components of the same type are housed in a single DIP when possible. This permits us to obtain a figure for the total area occupied by the IC's themselves; a good indication of the total area of a circuit board for a given multiplier relative to other multipliers.

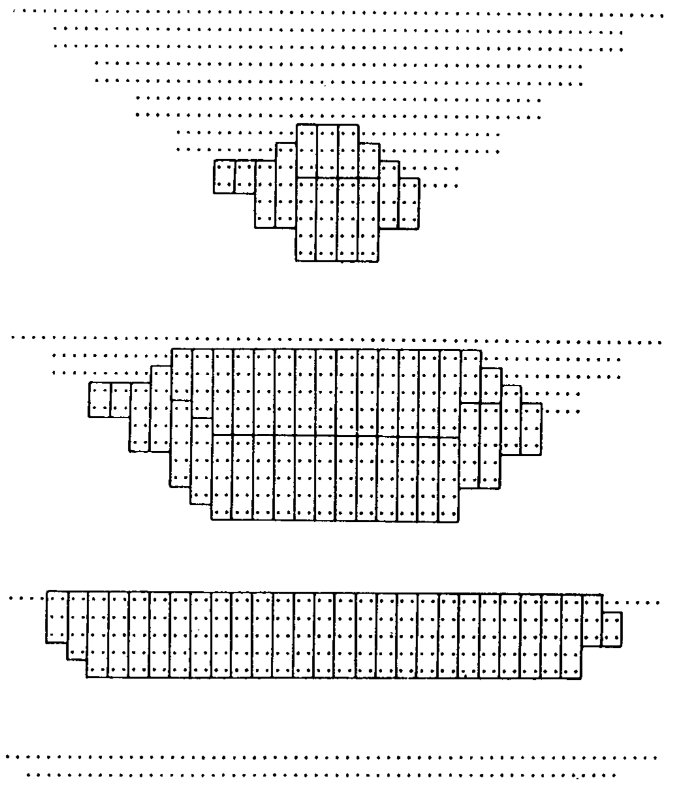

Fig. 7. 32 × 32-bit multiplication using only (5,5,4) counters.

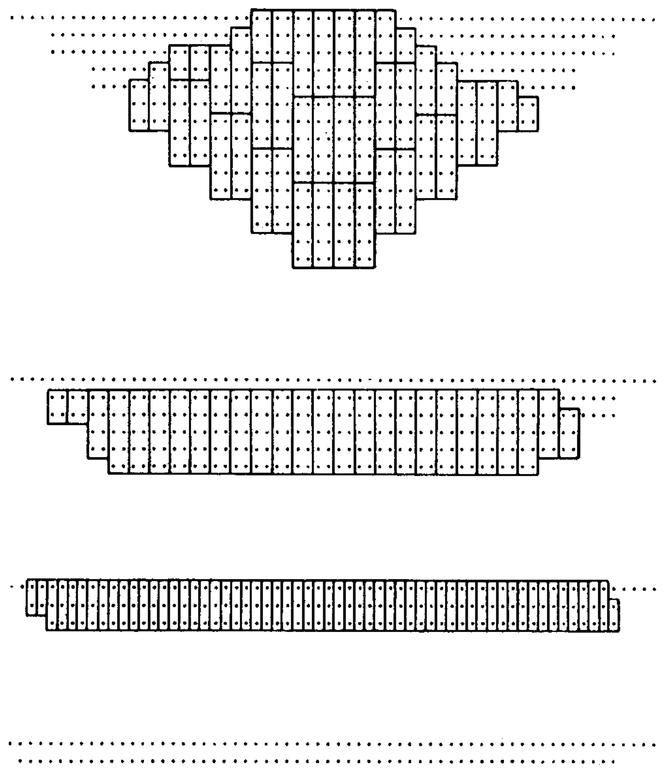

Fig. 8. 32 × 32-bit multiplication using (5,5,4) counters with (3,2) counters in last stage.

TABLE II
Characteristics of Some Multipliers and Counters

Counter or Multiplier	Logic Levels	# Per Pkg.	Pin Count	Package Area	Power in m.w.
(3,2)	3	2	14	.21	220
(7,3)	6	1	16	.24	650
(5,5,4)	6	1	16	.24	850
1 x 1	2	4	14	.21	113
4 x 4	6	1	20	.30	850

The characteristics of the various IC's are given in Table II. The cost data are not presented here since they fluctuate rapidly, but at the time of the comparison, in order of increasing cost, the rank was 3,1,2,4 independent of word length. Similarly, the rank, in order of increasing area, is 1,2,3,4 and, in order of increasing power dissipation, 3,1,2,4 independent of word length. Fig. 9 shows the result for propagation delay, for which the rank order does depend on word length.

Implementation of Multipliers and Counters

Multipliers and counters may be implemented directly by means of gates or by using read only memory. Direct implementation is feasible in counters only when the height of the input columns is small. Small units could be used as functional modules in constructing larger units by means of either LSI or hybrid techniques. This approach may also be applied to large multipliers [15].

Table lookup schemes are practical for $m \times m$ multipliers and larger counters. In its simplest form this type of scheme utilizes a standard ROM with one address bit corresponding to each input bit of the counter or multiplier and every input configuration addressing a distinct word in memory. This requires $2^i \times j$ bits of memory. Standard ROM's are attractive because the time and expense of designing and developing a new chip are avoided.

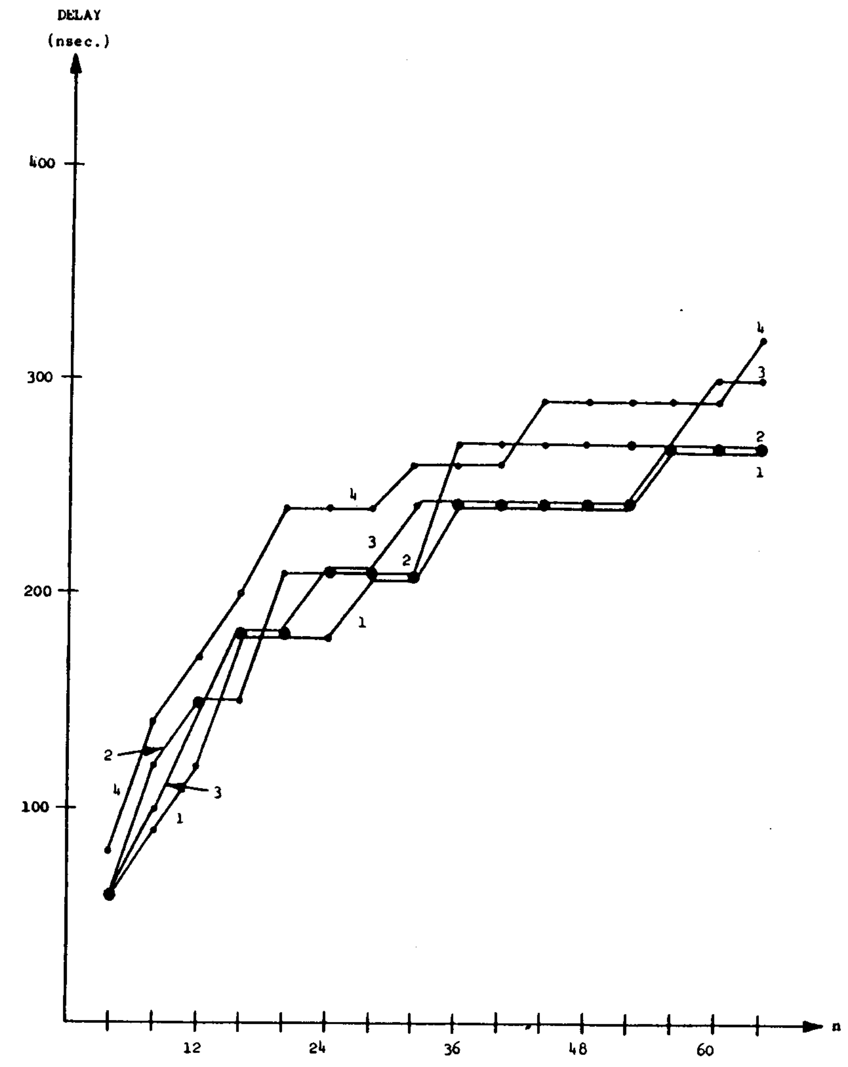

Fig. 9. Plot of propagation delay versus word size.

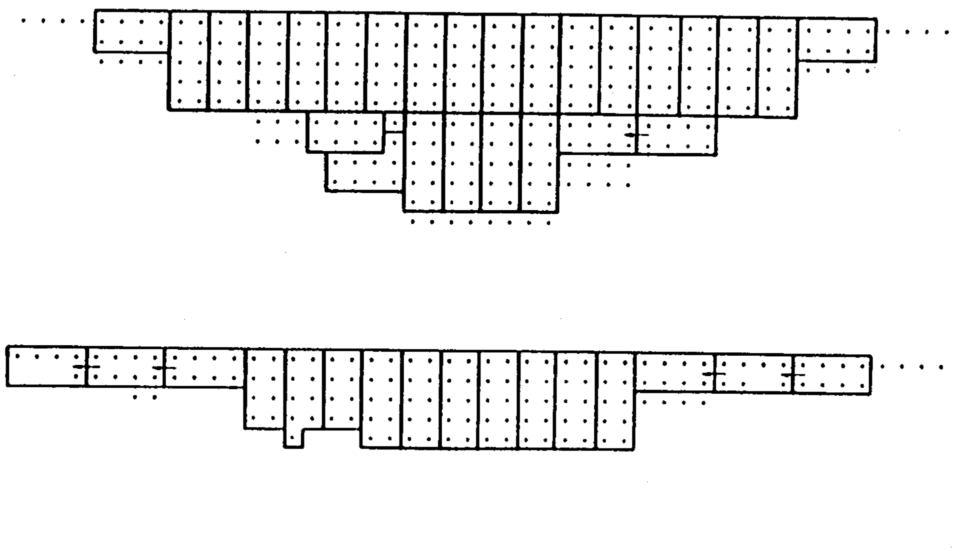

Fig. 10. 24 × 24-bit multiplier.

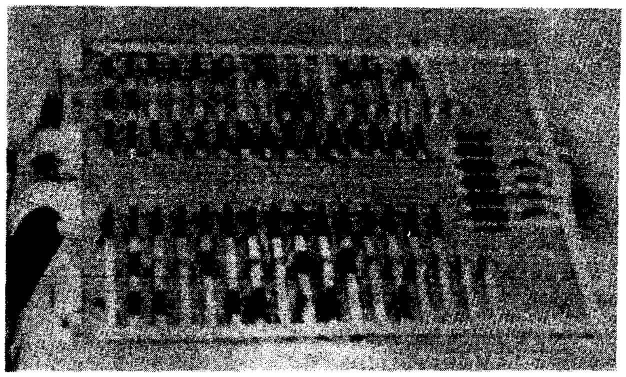

Fig. 11. Photograph of multiplier prototype.

The (intellectual) drawback of the standard ROM approach is the high degree of redundancy in the stored information. A reduction in the amount of storage required can be achieved if the input configurations are mapped into classes such that each member of a class references the same memory location. A scheme for such a mapping which incorporates residue threshold functions has been proposed by Ho and Chen [16].

The standard ROM implementation of a (5,5,4) counter requires 10 address bits and 4 output bits for a total of 4096 bits of memory. By removing the redundancies in the stored words the complete (5,5,4) counter can be programmed using 108 bits [14]. This represents a savings by a factor of nearly 40 over the 4096 bits required for direct implementation at the expense of one extra logic level in decoding. This type of implementation bears great similarity to programmed logic arrays (PLA's). In spite of the apparent economies of this approach, the ROM implementation may be more practical since large ROM's may always be more available than large PLA structures due to their greater generality.

Prototype Multiplier

A prototype 24 $\times$ 24-bit multiplier incorporating ROM implemented 4 $\times$ 4 multipliers and (5,5,4) counters as well as standard (2,2,2,3,5) counters was fabricated as a portion of a floating-point arithmetic unit [17]. The multiplier circuit was hand optimized to reduce the number of (5,5,4) counters and to minimize the propagation delay. The design is shown in Fig. 10. The carries from the (2,2,2,3,5) counters are propagated horizontally in some cases (indicated by arrows).

Fabrication limitations restricted us to two-sided boards with maximum dimensions of 15 $\times$ 18 in. The multiplier circuit contains 90 IC's, allowing three square inches of board area per package. Schottky TTL integrated circuits were used.

The layout of the board was done entirely by hand, the package placement being settled upon after several increasingly successful iterations. The board consists of a horseshoe-type arrangement of integrated circuits with the input lines running up the center of the horseshoe and the output lines running down the outside, as shown in Fig. 11. The thirty-six 4 $\times$ 4 multiplier ROM's immediately surround the input lines and are fed by horizontal connections on the back side of the board. The packages for the first level of reduction surround the 4 $\times$ 4 multiplier ROM's and are in turn surrounded by the second level and the carry look-ahead adders with their associated carry propagate units.

Both static and dynamic testing of the multiplier was performed. The ROM multipliers and counters were checked exhaustively and the overall multiplier was checked on numerous inputs.

For the dynamic tests the inputs and outputs were buffered through D-type registers. The propagation delay of the multiplier was measured by clearing the input register, clocking the data into the input register and, after a variable interval, clocking the output register. The variable interval was reduced until the outputs were no longer correct. Thus the input, output, and cable delays are included in the measured propagation delay. The measured propagation delay was 200 ns. The worst case calculated delay was 260 ns and the typical value was 190 ns.

Summary

The partial-product matrix generation-reduction schemes of Wallace and Dadda may be enhanced through the use of multiplier modules larger than 1 $\times$ 1 bit and counters larger and more general than (3,2) counters. The larger multiplier modules permit the generation of a partial-product matrix containing fewer total bits and having a maximum height less than that generated by 1 $\times$ 1 multipliers and, with current implementations, require fewer IC packages to do so. Larger, more complex counters permit reduction of matrices in fewer stages of counters with fewer IC packages than the simpler (3,2) counters while maintaining a similar number of total logic levels. The net result is a multiplier with the speed of Dadda's scheme and the compactness of current implementations of array multipliers [Advanced Micro Devices 25S05, Fairchild 9344].

Several large counters and multiplier modules have been realized as TTL integrated circuits and are also feasible as ECL circuits. The feasibility of their use in multiplication schemes has been demonstrated by the fabrication of a 200-ns 24 × 24-bit prototype. Advances in LSI and hybrid technology should make possible still larger counters and multiplier modules.

The algorithm for the logic design of multipliers incorporating generalized counters and multiplier modules is extremely straightforward. The physical implementation and maintenance of such multipliers is enhanced by their compactness and hence should require less time and expense than comparable multipliers employing Dadda's scheme, especially for large (e.g., 64 bits) words. This type of multiplier is attractive for a variety of applications, ranging from fast floating-point mini- and midicomputers to large scientific machines.

ACKNOWLEDGMENT

The authors wish to thank Dr. M. L. Graham and Prof. D. L. Slotnick for their suggestions and support during this work.

REFERENCES

[1] T. J. Chung and S. D. Bedrosian, "Fast digital multiplier based on iterative cellular arrays of ROMS," unpublished correspondence.
[2] I. D. Deegan, "Cellular multiplier for signed binary numbers," *Electron. Lett.*, vol. 7, pp. 436–437, July 29, 1971.
[3] J. C. Majithia and R. Kitai, "An iterative array for multiplication of signed binary numbers," *IEEE Trans. Comput.*, vol. C-20, pp. 214–216, Feb. 1971.
[4] S. D. Pezaris, "A 40 ns 17-bit array multiplier," *IEEE Trans. Comput.*, vol. C-20, pp. 442–447, Apr. 1971.
[5] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 19, pp. 349–356, May 1965.
[6] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Trans. Comput.*, vol. C-19, pp. 153–157, Feb. 1970.
[7] S. Singh and R. Waxman, "Multiple operand addition and multiplication," *IEEE Trans. Comput.*, vol. C-22, pp. 113–120, Feb. 1973.
[8] A. Svoboda, "Adder with distributed control," *IEEE Trans. Comput.*, vol. C-19, pp. 749–751, Aug. 1970.
[9] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, Feb. 1964.
[10] A. R. Meo, "Arithmetic networks and their minimization using a new line of elementary units," *IEEE Trans. Comput.*, vol. C-24, pp. 258–280, Mar. 1975.
[11] J. A. Gibson and R. W. Gibbard, "Synthesis and comparison of two's complement parallel multipliers," *IEEE Trans. Comput.*, vol. C-24, pp. 1020–1027, Oct. 1975.
[12] N. G. Kingsbury, "High speed binary multiplier," *Electron. Lett.*, vol. 7, pp. 277–278, May 20, 1971.
[13] C. R. Baugh and B. A. Wooley, "Two's complement parallel array multiplication algorithm," *IEEE Trans. Comput.*, vol. C-22, pp. 1045–1047, Dec. 1973.
[14] W. J. Stenzel, "A class of compact high speed parallel multiplication schemes," Univ. of Illinois at Urbana-Champaign, Dep. Comput. Sci., Rep. UIUCDCS-R-75-756.
[15] D. R Breuer, "A high speed monolithic 8 × 8 multiplier," unpublished correspondence.
[16] I. T. Ho and T. C. Chen, "Multiple addition by residue threshold functions and their representation by array logic," *IEEE Trans. Comput.*, vol. C-22, pp. 762–767, Aug. 1973.
[17] M. L. Graham and D. L. Slotnick, "An array computer for the class of problems typified by the general circulation model of the atmosphere," Univ. of Illinois at Urbana-Champaign, Dep. Comput. Sci. Rep. UIUCDCS-R-75-761.