1-1-1986

Algorithms for Iterative Array Multiplication

Shinji Nakamura
Dartmouth College

ALGORITHMS FOR ITERATIVE ARRAY MULTIPLICATION

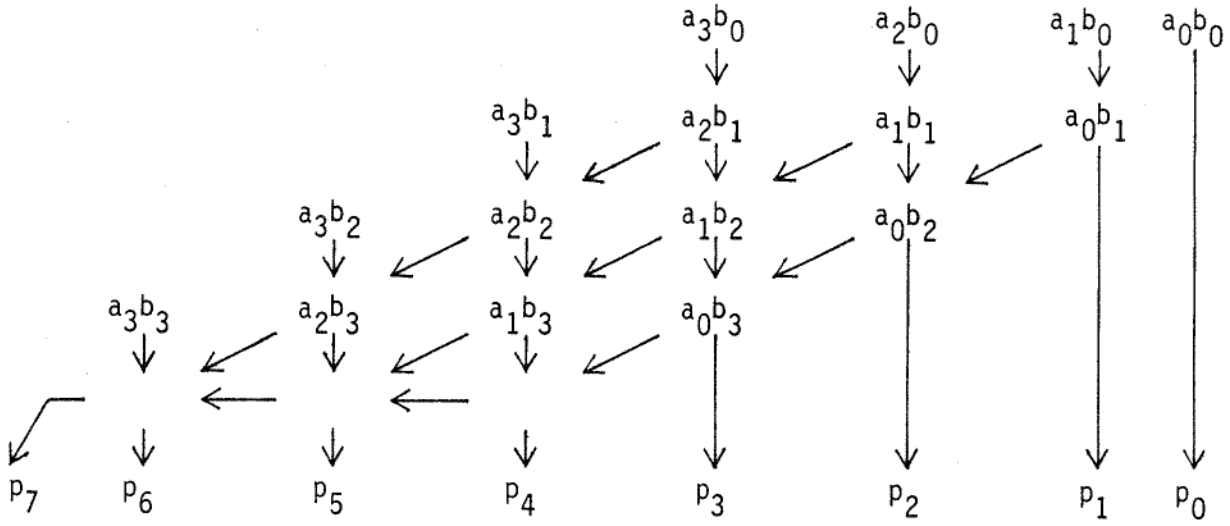
Shinji Nakamura

Technical Report PCS-TR86-106

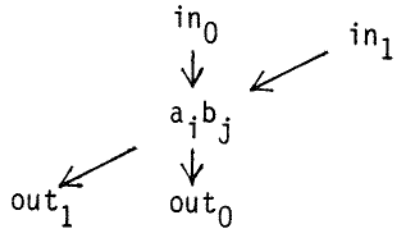
1. Introduction

The multiplication operation is one of the most vital functions in many computer applications. Because of the inherent complexity of the operation, the execution speed of multiplication tends to be the dominant factor in the entire processing time. Therefore, with the advent of VLSI technology, parallel algorithms for multiplication become increasingly important. In this paper we will discuss new multipliers which are composed of identical cells interconnected in a homogeneous connection pattern. The speed of the multiplication algorithms can be twice as fast as known algorithms for the same type of multipliers without substantially increasing the complexity of the hardware. Since the interconnections required by the algorithms are realized in systematic structures, the algorithms are ideal for VLSI implementation.

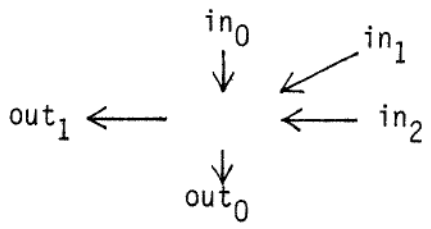
With our usual numbering system the multiplication operation of two n-digit numbers is based on n^2 one digit by one digit multiplications, and n additions of n digit numbers, which are appropriately shifted to the correct positions. The same holds for the binary system. However, since the multiplication of one binary digit, a_i, by another binary digit, b_j, is just a simple logical AND operation between a_i and b_j, the major part of the multiplication is in the n additions of n bit binary numbers. In ordinary sequential multiplication the multiplicand is shifted bit by bit and added to a large 2n-bit accumulator when the bit at the corresponding bit position of the multiplier is 1. To increase the multiplication speed the addition operation can be performed in parallel. In a straightforward parallel multiplication [1], the addition operations are carried out by an array of n(n-1) full adders interconnected as shown in Figure 1(a). In the figure full adders are



(a) A 4 by 4 bit binary full adder multiplier.



(b) A full adder cell with a partial-product a_ib_j as an input. The output value is calculated as: $2out_1+out_0 = in_0+in_1+a_ib_j$.



(c) A full adder cell without partial product. The output value is calculated as: $2out_1+out_0 = in_0+in_1+in_2$.

Figure 1

shown using the diagrams shown in Figure 1(b), (c), or a variation of these diagrams; where input arrows are removed the values of removed inputs are assumed to be zero. After generating n^2 bit partial-products of $a_i b_j (0 \leq i,j \leq n)$, the full adder array adds the partial results simultaneously. This method is referred to as an iterative array algorithm, in which the same type of component cells, or simply cells, are interconnected by a structured connection pattern. We refer to this particular iterative array algorithm as a straightforward algorithm, or simply a binary full adder multiplier. The speed of multiplication is expressed by the delay time associated with the array cells. For the above straightforward algorithm the speed of one n bit by n bit multiplication is 2n-1 cell delays since the longest carry propagate chain in the array is through the right and bottom edge of the parallelogram composed of full adder cells. The number of cell delays for the parallel multiplication can be decreased substantially by applying another type of method which allows more freedom in the variety of components and their interconnections [2]-[4]. This approach, referred to as a partial-product matrix reduction or column compression scheme, is based on the use of various parallel counters and can make the number of cell delays 0(log(n)) to 0(log...log(n)) for an n bit by n bit multiplication. Compared to the linear delay of the iterative array algorithm, column compression is more advantageous as to the number of cell delays, especially for large n. For VLSI implementations of a single chip multiplier, however, column compression, requiring a large number of irregular interconnections between different types of cells, is at a major disadvantage over the iterative array algorithm. In other words, column compression is advantageous for multiple chip multipliers where the wiring between chips is not a critical issue.

The single chip implementation of a multiplier is very attractive in many respects. It is very compact and can be a built-in multiplier for a single chip processor, but most of all, since a single chip multiplier does not have to transfer the intermediate results between chips, there is no overhead for the amplifiers which drive off-chip devices. Thus the cell delay is much smaller than those cell delays associated with the multiple chip cells. In a single chip design the interconnections between cells become very significant. In many cases it is the dominant factor of the design because wiring must be done with two-dimensional restrictions and consumes area which could otherwise be used for logic. From these facts, although the number of cell delays tends to be greater, iterative array algorithms are more suitable for VLSI parallel array multipliers. Therefore, we only consider iterative array algorithms in subsequent discussions.

The above-mentioned iterative array multiplier of Figure 1 was proposed for unsigned integer multiplication, later 2's complement array multiplication algorithms were introduced [5], [6], but their structure of the array and interconnection between the array elements are almost identical to the original design, and the number of full adders and delay times of the two's complement multipliers differ only by constant factors from the unsigned one (i.e., the difference is independent of the length of multiplicand and multiplier). For these highly regular iterative array multipliers, as far as the basic structure and multiplication speed is concerned, there has been no major improvement since the original unsigned integer array multiplier was introduced. In this paper we discuss two new iterative array multiplication algorithms, based on different types of cells, and their interconnection structures, which result in faster multiplication speed.

In Section 2, we discuss a simple speedup method for the straightforward algorithm of Figure 1, which uses higher radix. Our new algorithms are presented in Sections 3 and 4 and their speed and complexity are discussed using the higher radix method as a benchmark.

2. Radix 4 Multiplier

One obvious method to improve the speed of the original full adder array multiplier while using a similar interconnection structure between cells is to adopt a higher radix numbering system than the binary system. For instance, the same multiplication as that of the binary 4 by 4 array multiplier of Figure 1 can be realized by a 2 by 2 array multiplier of radix 4. The two 4 bit numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ are expressed by A_1A_0 and B_1B_0 where A_i and B_j represent one digit of radix 4 number. The radix 4 multiplier can be composed of radix 4 full adders as shown in Figure 2. Therefore if we implement a radix 4 full adder which can perform three (in general four but for small size arrays such as 2 by 2 it can be done in three) radix 4 digit additions at binary full adder speed, the radix 4 array multiplier is approximately twice as fast as the original 4 by 4 binary array multiplier. In general, to perform the equivalent of an n by n binary multiplication using a radix 2^r array multiplier, approximately $2^{-2(r-1)}n^2$ full adders are required and the multiplication speed is $2^{-(r-1)}n$ full adder delays. That is, by assuming that all full adders have the same delay time independent of the radix, incrementing the radix by one reduces the number of full adders by 1/4 and the multiplication time by 1/2. Hence, if the complexity of a radix 4 full adder is about four times that of a binary full adder we would have a multiplier which is twice as fast as the binary multiplier without increasing its total complexity. To make this argument more precise let's consider the radix 4 multiplier in more detail.

The radix 4 partial product A_iB_j produces a two digit result, P_{ij} and Q_{ij}, with the positional weight of 4^{i+j+1} and 4^{i+j}, respectively. The ij^{th} cell in the radix 4 multiplier is a radix 4 full adder. Two of its inputs, Q_{ij} and $P_{i-1,j}$, are halves of the partial products A_iB_j and $A_{i-1}B_j$. The other inputs to the ij cell are supplied from its neighboring cells so that the inter-

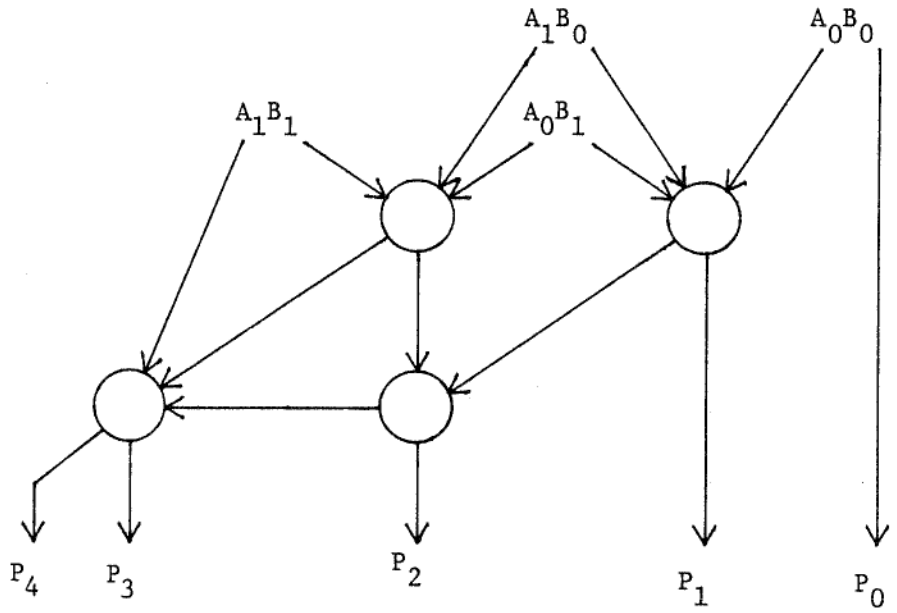


Figure 2. A 2 by 2 radix 4 multiplier

connection between the ij cell and its neighbor should be as shown in Figure 3. As seen from Figure 3 the cell must be a radix 4 adder of four one-digit inputs. Since in electronic digital circuits the direct realization of radix 4 numbers by four distinct levels of voltage potential or current flow is hardly practical, a reasonable design for the cell is obtained by representing one radix 4 digit by two binary positions. One such realization is shown in Figure 4(b).

In the diagram the A, B, Q, S, C, and P radix 4 inputs are replaced by a_1a_0, b_1b_0, q_1q_0, s_1s_0, c_1c_0, and p_1p_0 two bit binary inputs and the output C and S are replaced by c_1c_0 and s_1s_0 binary outputs. A comparison of the radix 4 cell with a radix 2 cell (Figure 4(a)) makes it clear that the radix 4 is far more than four times as complex as the radix 2 cell. The number of input lines has increased from four to ten, the outputs from two to four, one single AND gate was replaced by one 2 bit by 2 bit multiplier, and a full adder was substituted by an adder of four two bit numbers which calculates

$$2(q_1 + s_1 + c_1 + p_1) + q_0 + s_0 + c_0 + p_0$$

and produces their four bit sum. To gain the improvement in the multiplication speed by the factor of two, the four two-bit addition must be done in one full adder delay time. To make a quantitative argument for this comparison we introduce the complexity measure that was used for the general counters [3] in the column compression technique. Column compression is accomplished by various parallel counters which count the number of partial products whose value is one and have the same weight. The implementation of parallel counters, even for a moderate number of inputs such as 7, by a combinational circuit of small cascading levels is difficult because the number of inputs for a gate becomes very large. Therefore, a practical implementation of such a high-speed counter is based on ROM memory where the counting is done by a table

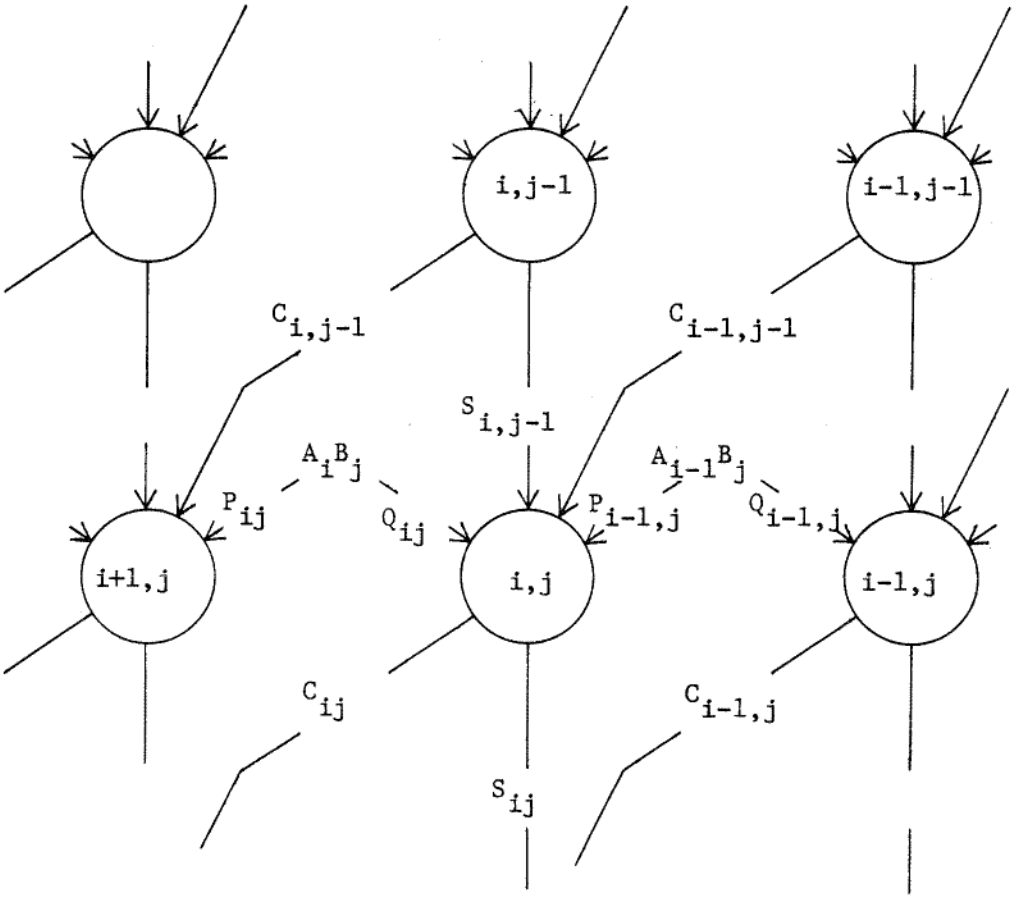
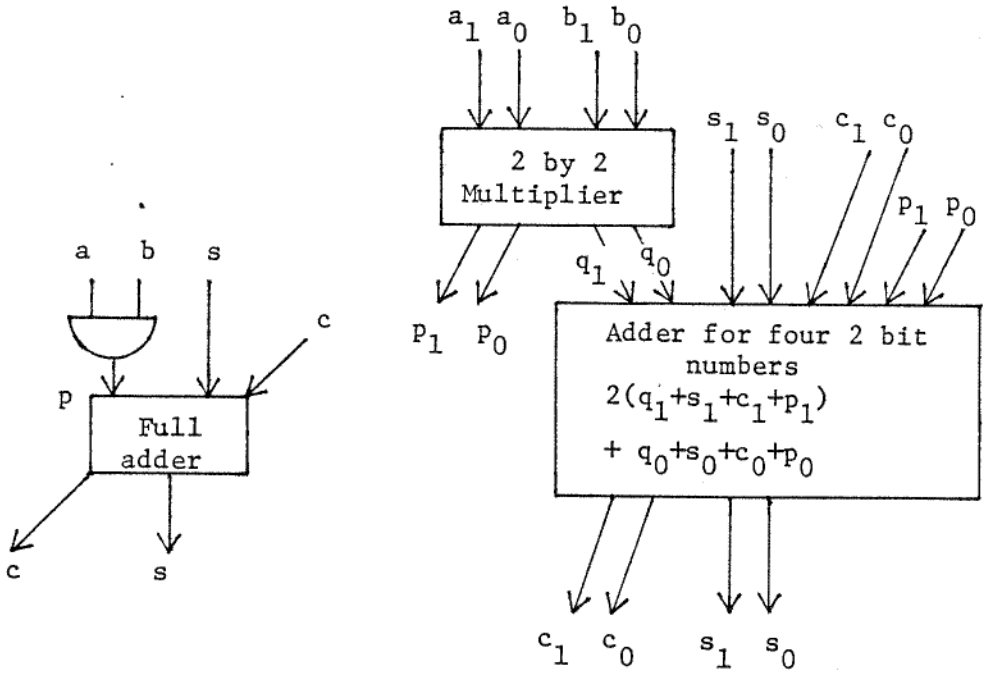


Figure 3. The interconnection of radix 4 full adders. A circle indicates an adder of four radix 4 digits whose position in the array is shown by the indices in the circle.



(a) A radix 2 cell

(b) A radix 4 cell

Figure 4.

lookup operation on the ROM memory. The number of ROM memory bits necessary for a counter gives a clear measure of the hardware complexity of the counter. For instance, a 7-counter has 7 input lines and 3 output lines. Each output requires 2^7 ROM bits to produce the results for all possible input values, so that a 7-counter requires a total of $2^7 \times 3 = 384$ ROM bits. The same practical difficulty exists for the radix 4 adder cell; it has too many inputs to realize the cell in a flat combinational circuit. Since A_i and B_j are primary inputs and their values are defined from the beginning, the one digit radix 4 multiplier which generates a radix 4 partial product can be designed in a many-level combinational circuit, but the rest of the cell still has 8 inputs. If we adopt the same table lookup method for the radix 4 cell, the complexity of one cell, excluding the one digit radix 4 multiplier, becomes $2^8 \times 4 = 1024$ bits because there are 8 inputs and 4 outputs. For the entire multiplier there are $(1/4)n^2$ cells so that the complexity is $1024 \times (1/4)n^2 = 256n^2$. Note that this value does not include the complexity of the one digit radix 4 multipliers, so that the actual complexity is greater than this value.

Although the complexity measure discussed is based on a specific design method the results represent the relative complexities of combinational designs when the number of gate levels are small. The binary full adder multiplier has the cell complexity of $2^3 \times 2 = 16$ and a total complexity of $16n^2$. Hence, by this measure it is concluded that the radix 4 algorithm increased the speed by a factor of two and at the same time increased the hardware complexity by a factor of 16 (which probably actually reduced the speed!). It is clear that the required complexity is too heavy for the gain in speed. This is why the higher radix number systems are not popular in the actual design of array multipliers.

Then the question we address is, is there any other iterative array

algorithm which may have more cells than the radix 4 multiplier with a different interconnection structure between them, but that would still have a faster multiplication speed, faster than the 2n binary full adder equivalent delay time? Our affirmative answer to this question are the two new array multipliers described in the following sections.

3. Iterative Array Multiplication Using 5-Counters

Our first new iterative array algorithm is based on the five input parallel counter. The five input counter, or simply the 5-counter, is expressed by the diagram shown in Figure 5. The five inputs to the 5-counter are the three inputs through input lines in_0, in_{-1}, in_{-2} and the two partial products of a_ib_j and a_jb_i, which are shown at the center of the diagram. The 5-counter counts the number of ones on these 5 inputs and produces a binary number of the count on the three output lines out_0, out_1, and out_2, which correspond to the 2^0, 2^1, and 2^2 positions, respectively, i.e., the function of a 5-counter cell is:

$$out_2 2^2 + out_1 2^1 + out_0 = in_0 + in_{-1} + in_{-2} + a_ib_j + a_jb_i$$

The entire interconnection of the input and output lines of 5-counter cells in a 6 bit by 6 bit multiplier of two numbers

$$a_5a_4a_3a_2a_1a_0 \times b_5b_4b_3b_2b_1b_0 = p_{11}p_{10}\cdots p_0,$$

is shown in Figure 6. We refer to this iterative array algorithm as a 5-counter multiplier.

In Figure 6 the numbering of rows and columns starts at the bottom row and leftmost column with 0. The cell at (i,j) position, C_{ij}, has a,b inputs of a_ib_j and a_jb_i for $i,j<5$. The cells in the top row marked as "D" are delay cells to synchronize the carry signals. If the circuit is designed by asynchronous combinational logic, these delay cells can be replaced by simple wires. The small cells on the diagonal marked by "⊕" are exclusive-OR gates.

In general, for n+1 bit by n+1 bit multiplier of

$$a_na_{n-1}\cdots a_0 \times b_nb_{n-1}\cdots b_0 = p_{2n+1}p_{2n}\cdots p_0,$$

has interconnections defined as follows:

(1) For $i,j>0$ and $i+j<n$, C_{ij} has inputs from product terms a_ib_j and a_jb_i. The in_0, in_{-1}, and in_{-2} input lines are connected to $C_{i+1,j-1}$, $C_{i,j-1}$,

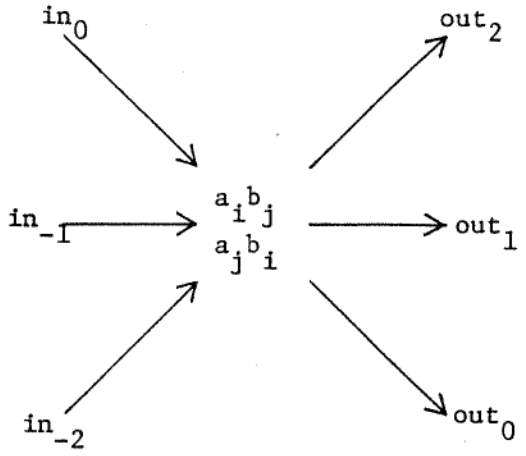


Figure 5. A 5-counter cell. The output value is calculated as:

$$\sum_{i=0}^{2} out_i 2^i = a_i b_j + a_j b_i + \sum_{i=-2}^{0} in_i.$$

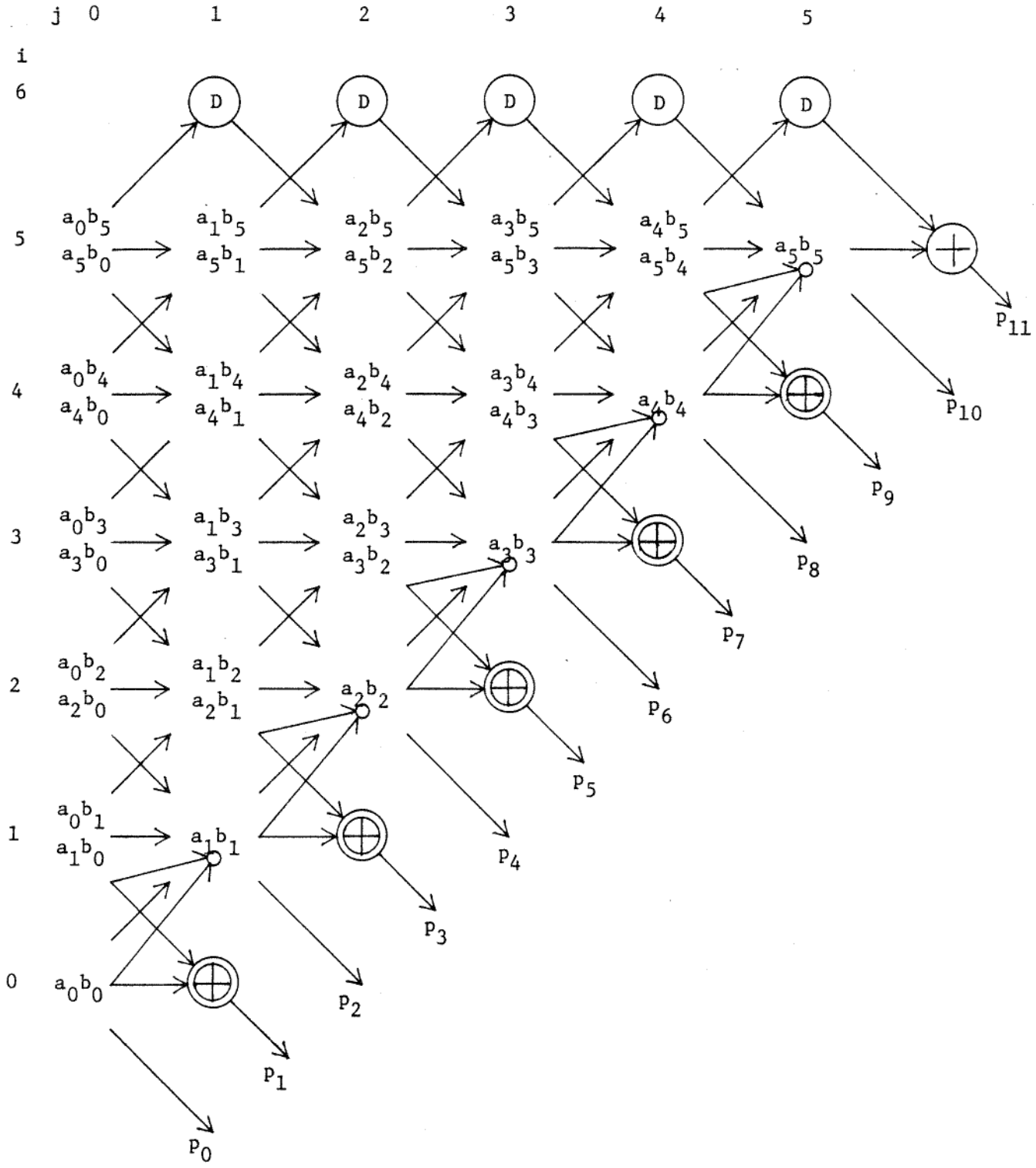


Figure 6. A 6 bit by 6 bit 5-counter multiplier. ⊕ indicates an exclusive OR and ⊕ indicates an inclusive OR.

and $C_{i-1,j-1}$ cells through their out_0, out_1, and out_2 output lines, respectively. The out_2, out_1, and out_0 output lines of C_{ij} are connected to the $C_{i+1,j+1}$, $C_{i,j+1}$, and $C_{i-1,j+1}$ cells through their input lines in_{-2}, in_{-1}, and in_0, respectively.

(2) The diagonal cells C_{ij}, where i+j=n, have only one input from product terms $a_j b_j$. The other product term is produced by an AND operation between the out_0 output of the $C_{i,j-1}$ cell and the out_1 output of the $C_{i-1,j-1}$ cell. The same out_0 and out_1 outputs from the $C_{i,j-1}$ and $C_{i-1,j-1}$ cells are connected to an exclusive OR gate to produce the 2j-1 bit of the result, i.e., p_{2j-1}. The out_0 output line of the diagonal cell C_{jj} is the $2j^{th}$ bit of the result, i.e., p_{2j}.

(3) The output line out_2 of the cell $C_{0,j}$ in the second row from the top, j<n, is connected to the delay cell $D_{n,j+1}$. The input line in_0 of $C_{0,j}$ is connected to the output line of the delay cell $D_{n,j-1}$.

(4) The input lines in_0, in_{-1}, and in_{-2} of the leftmost column's cells, $C_{i,j}$, are connected to the 0.

As shown in Figure 6 the carry propagates from left to right so that the longest delay for n bit by n bit multiplication is n 5-counter delays. The diagonal cells have a slight difference from the rest of the cells. Since one of their product inputs is not produced directly from the primary inputs of the a's and b's but from the output lines of the previous stage cells, the direct implementation by an AND gate will increase the one cell delay by an additional one AND gate delay. Hence, the total delay is increased by n AND gate delays. However by applying some acceleration method to these diagonal cells, one AND gate delay per one cell could be absorbed without significantly increasing the hardware complexity. An alternative solution for this problem is to leave the diagonal cells as 3 input counters and add another stage of n full adders, each

of which have four inputs. Two of the four inputs have a positional weight of 1/2 so the SUM and CARRY outputs of the full adder cells C_{ij} are defined as:

$$out_0 = SUM(w,x,y,z) = w \oplus x \oplus (yz)$$

$$= wxyz + w\bar{x}\bar{y} + w\bar{x}\bar{z} + \bar{w}x\bar{y} + \bar{w}x\bar{z} + \bar{w}\bar{x}yz,$$

$$out_1 = CARRY(w,x,y,z) = wx + wyz + xyz,$$

where $w = out_0$ of $C_{i+1,j-1}$, $x = out_1$ of $C_{i,j-1}$, $y = out_0$ of $C_{i,j-1}$, and $z = out_1$ of $C_{i-1,j-1}$; y,z inputs have the positional weight of 1/2. Such a full adder cell is simpler and faster than a 5-counter. By the alternative solution, overall delay is n 5-counters plus one full adder delay. Some minor improvements in speed can be made by replacing all the 0^{th} column cells (i.e., $C_{i,0}$) by half adders.

The major issue in this array multiplication algorithm is the complexity of a single 5-counter cell and its delay time. Since the longest carry chain in the array is n for n bit multiplication, if the delay speed of a single 5-counter is equal to a binary full adder delay the new algorithm is twice as fast as the old one. But such a 5-counter should be more than twice as complex as a binary full adder - but how much more? According to the same complexity measure we used for the radix 4 multiplier, the complexity of a 5-counter is expressed as $2^5 \times 3 = 96$. This value is six times more than that of a binary full adder, but because the required number of 5-counters in a multiplier is approximately $(1/2)n^2$, the 5-counter multiplier is three times as complex as the binary full adder multiplier. Hence, with the new iterative array algorithm the delay time reduces to 1/2 by sacrificing the hardware complexity by the factor of three.

The other critical aspect in the comparison is their requirement of interconnections. In this respect, the 5-counter multiplier is very close to the binary full adder multiplier, because only one more interconnection line per

cell is needed for the 5-counter multiplier. The added line is used to transmit a super carry which propagates twice as fast as the ordinary carry. This is why the 5-counter multiplier has an n cell delay rather than 2n. As shown in Figure 6 all the interconnections between cells are made between either nearest neighbors or next-nearest neighbors. This property is vital for VLSI implementation because wiring in integrated circuits is confined to two-dimensional restriction.

4. Iterative Array Multiplier by Generalized (6,3,4) Counters

Our second algorithm with n cell delay speed is based on two different types of cells and a nearly regular, but partially irregular, interconnection structure of the cells. Therefore, in the strict sense this design is not an iterative array algorithm; however, as is shown in Figure 8 the irregular part is limited to a diagonal column at the final output stage and the design is still practical for VLSI single chip implementation.

The notion of a generalized $(c_{k-1},c_{k-2},\ldots c_0,d)$ counter [4] which has $\sum_{i=0}^{k} c_i$ inputs and d outputs was extended from the notion of an ordinary counter [3]. The output value v of a $(c_{k-1},c_{k-2},\ldots c_0,d)$ counter depends on the weighted binary inputs in_{ij}, where there are c_i bits of binary inputs that have the weight of 2^i. Hence the value v is:

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

Since d is the number of output lines from the counter, the following inequality should hold:

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

With this generalized notation of a counter, the 5-counter and the binary full adder we discussed in the previous sections can be expressed as a (5,3) counter, and (3,2) counter, respectively.

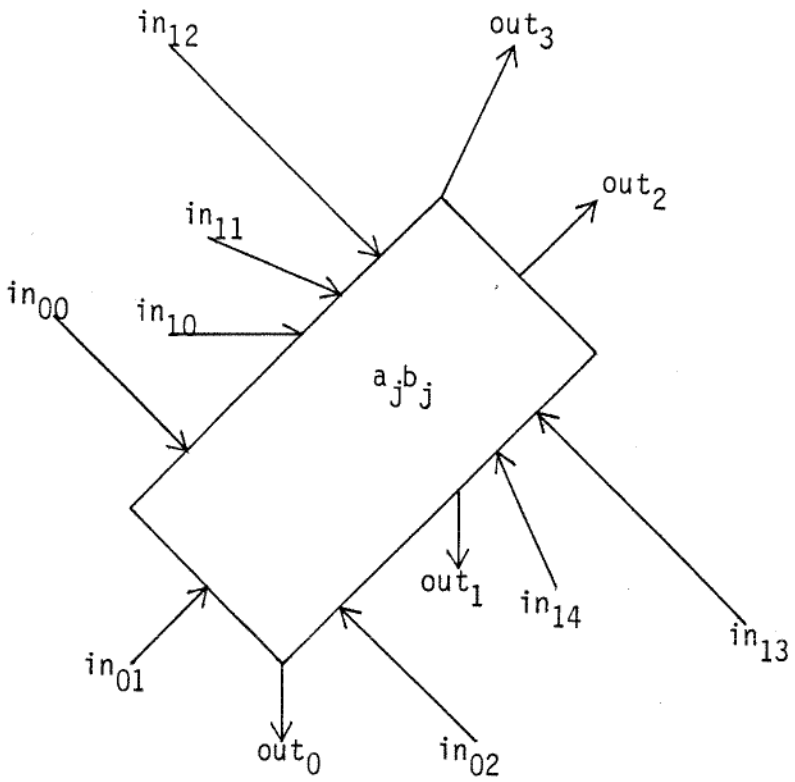
The new iterative array multiplier uses n-1 (6,3,4) counters and (n-2)(n-3) full adders. We refer to this iterative array algorithm as a (6,3,4) counter multiplier. A (6,3,4) counter has a total of nine inputs, of which six are weighted by 2^1 and the other three by 2^0. Therefore, the maximum value of the four output lines of the counter is $6\times2^1 + 3\times2^0 = 15$. The

graphical diagram of a (6,3,4) counter is shown in Figure 7. The six inputs with a weight of 2^1 are in_{10} through in_{14} and a partial product $a_j b_j$, and the three inputs with a weight of 2^0 are in_{00}, in_{01}, and in_{02}. The four outputs o_i, $0 \leq i \leq 3$, have a weight of 2^i; i.e., the function of a (6,3,4) counter is:

$$\sum_{i=0}^{3} out_i 2^i = 2\left(a_j b_j + \sum_{i=0}^{4} in_{1i}\right) + \sum_{i=0}^{2} in_{0i}$$

A 6 bit by 6 bit multiplier with this array algorithm is shown in Figure 8. The n-1 (6,3,4) counters are placed in a main diagonal column of the array and the (n-2)(n-3) full adders, expressed by similar diagrams of Figure 1(b), are placed on the grid points in the rest of the array. Through the binary full adders both sums and carrys propagate in two directions. Sums propagate diagonally from upper left to lower right and lower right to upper left. Carrys propagate from left to right and bottom to top. All the four streams of sums and carrys meet at the main diagonal where the (6,3,4) counter chain is located. The (6,3,4) counter chain absorbs all the partial results from the four streams of data flow and produces the final 2n-bit results in an n cell delay time. Since the longest carry chains in the array are n the entire speed is also n (complex) cell delays. In the previous 5-counter multiplier all the cells have a super carry line, and the extra task of speeding up the operation is evenly distributed across the entire array, while in the new algorithm the same result is achieved by changing the flow of data streams and speeding up the process in the data congested area by introducing heavily equipped super cells, i.e., (6,3,4) counters.

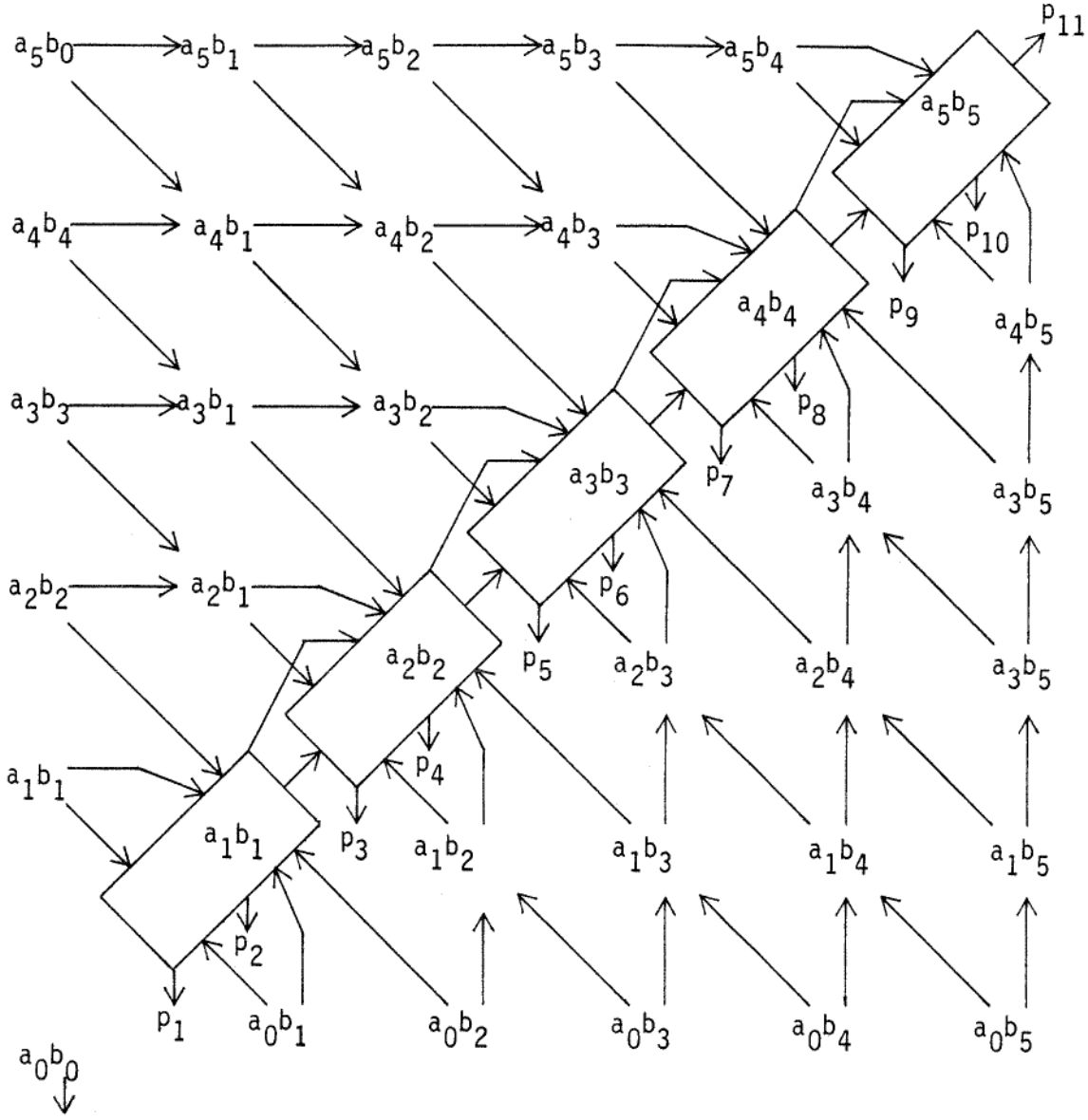
By the same complexity measure used in the previous sections, a (6,3,4) counter is $2^9 \times 4 = 2048$, which is two times more complex than a radix 4 full adder and more than 20 times as complex as a 5-counter. A comparison of complexities in 6 by 6 bit array multipliers by the four different algorithms



A (6,3,4) counter cell. The output value is calculated as:

$$\sum_{i=0}^{3} out_i 2^i = 2\left(a_j b_j + \sum_{i=0}^{4} in_{1i}\right) + \sum_{i=0}^{2} in_{0i}$$

Figure 7



A 6 by 6 bit (6,3,4) counter multiplier.

Figure 8

upon which they are based, binary full adder, radix 4 full adder, 5-counter, and (6,3,4) counter, is shown in Table 1.

Algorithm	Complexity in Number of ROM Memory Bit	Normalized
Binary full adder	424	1.00
Radix 4	4736	11.17
5-counter	956	2.25
(6,3,4) counter	6480	15.28

Table 1. Complexities of 6 by 6 bit array multipliers by four different iterative array algorithms.

In Table 1 all the absolute complexity values were calculated after applying possible simplifications for the cells near boundaries. Because of the peculiarities of the small value of n=6 the normalized complexities for the radix 4 and 5-counter cases are smaller than the predicted values of 16 and 3, respectively. With increased values of n, since the number of internal cells becomes dominant, the normalized complexities asymptotically approach 16 and 3.

The most complicated algorithm in the table is the (6,3,4) counter multiplier, which is 15.28 times as complex as the binary full adder multiplier, but it should be noted that this is only for n=6 or for small n. The number of (6,3,4) counters in an n by n bit multiplier is n-1 while the number of binary full adders is (n-2)(n-3) so that when n increases, the algorithm's overall complexity is dominated by the binary full adders. Therefore, the normalized complexity of the (6,3,4) multiplier is asymptotically equal to 1. If we exclude the requirements in interconnection structure,

when n=50 the complexity of a (6,3,4) counter multiplier is approximately equal to that of a 5-counter multiplier. For n>50 the (6,3,4) counter multiplier becomes simpler than the 5-counter multiplier.

Even though the (6,3,4) counter multiplier has a homogeneous interconnection structure among the binary full adder cells, the diagonal (6,3,4) counter cells introduce substantial irregularity in the array structure. From the complexity measure a single (6,3,4) counter needs 128 times more area than a binary full adder cell so that the counter chain cannot be embedded in an array as shown in Figure 8. Hence, the practical VLSI single chip design of this algorithm will have two separate segments, one for the (n-2)(n-3) full adder segment and the other for the n-1 (6,3,4) counter segment, and approximately 4n distinct wires, which will route partial results from the full adder segment to the (6,3,4) counter segment. The previous complexity comparison made for the 6 by 6 bit multipliers did not include this factor so that the actual value for the (6,3,4) counter multiplier will actually be much larger. This is also true for the comparison between the (6,3,4) counter multiplier and the 5-counter multiplier. Hence, the break-even size, n=50, will increase significantly; however, since the dominant factor is still the binary full adder segment and not the interconnection between the two segments, when n becomes sufficiently large the (6,3,4) counter multiplier will surpass the 5-counter multiplier and approach the binary full adder multiplier in the hardware complexity.

5. Summary of Comparisons

For the four iterative array multiplication algorithms we measured their speed by the number of cell delays, with the assumption that, irrespective of their functions, any type of cell had the same operational speed of one cell delay. Because of this assumption, we measured the complexity of a single cell by the number of ROM memory bits necessary to implement the cell by the table lookup operation on the ROM memory. The complexity of an algorithm was calculated by summing up the complexity of all cells used in the algorithm. The speeds and complexities of the four algorithms are summarized in Table 2.

Algorithm	Speed in Number of Cell Delays	Complexity in Number of ROM Bits	Normalized Asymptotic Complexity
Binary full adder	2n	$16n^2-16n$	1
Radix 4	n	$256n^2$	16
5-counter	n	$48n^2$	3
(6,3,4) counter	n	$16n^2+1968n-1952$	1

Table 2. Speed and Complexity of Iterative Array Multiplication Algorithms.

The first three algorithms, binary full adder, radix 4, and 5-counter multipliers have a homogeneous iterative array structure. All of the three require cell interconnections, either with the nearest or next nearest neighbor cells. A binary full adder cell has 3 inputs and 2 outputs and a 5-counter cell has 5 inputs and 3 outputs, but the total number of 5-counter cells required in a 5-counter multiplier is about half that of a binary full adder

multiplier. Hence, their interconnection complexity is approximately the same. Although the interconnection structure of the (6,3,4) counter multiplier is much more complex than the others, for sufficiently large n, this algorithm is the best in terms of the speed and hardware complexity; however for reasonable sizes of n (at least $n<50$), the 5-counter multiplier is simpler than the (6,3,4) counter multiplier.

For those multiplication algorithms which are based on the binary number system, the conversion from unsigned multiplication to signed two's complement multiplication is simple. For instance, by applying the usual technique of two's complement number addition [6], a two's complement 5-counter multiplier can be realized simply by changing some of the AND circuits to NAND and the final output OR gate to a NOR gate. A 6 by 6 bit two's complement multiplier is shown in Figure 9. A similar type of conversion can be made for the binary full adder and (6,3,4) counter multipliers. However, the above technique is only for the binary number system; the signed radix 4 multiplier requires some other method (e.g., sign conversion) which would increase hardware complexity.

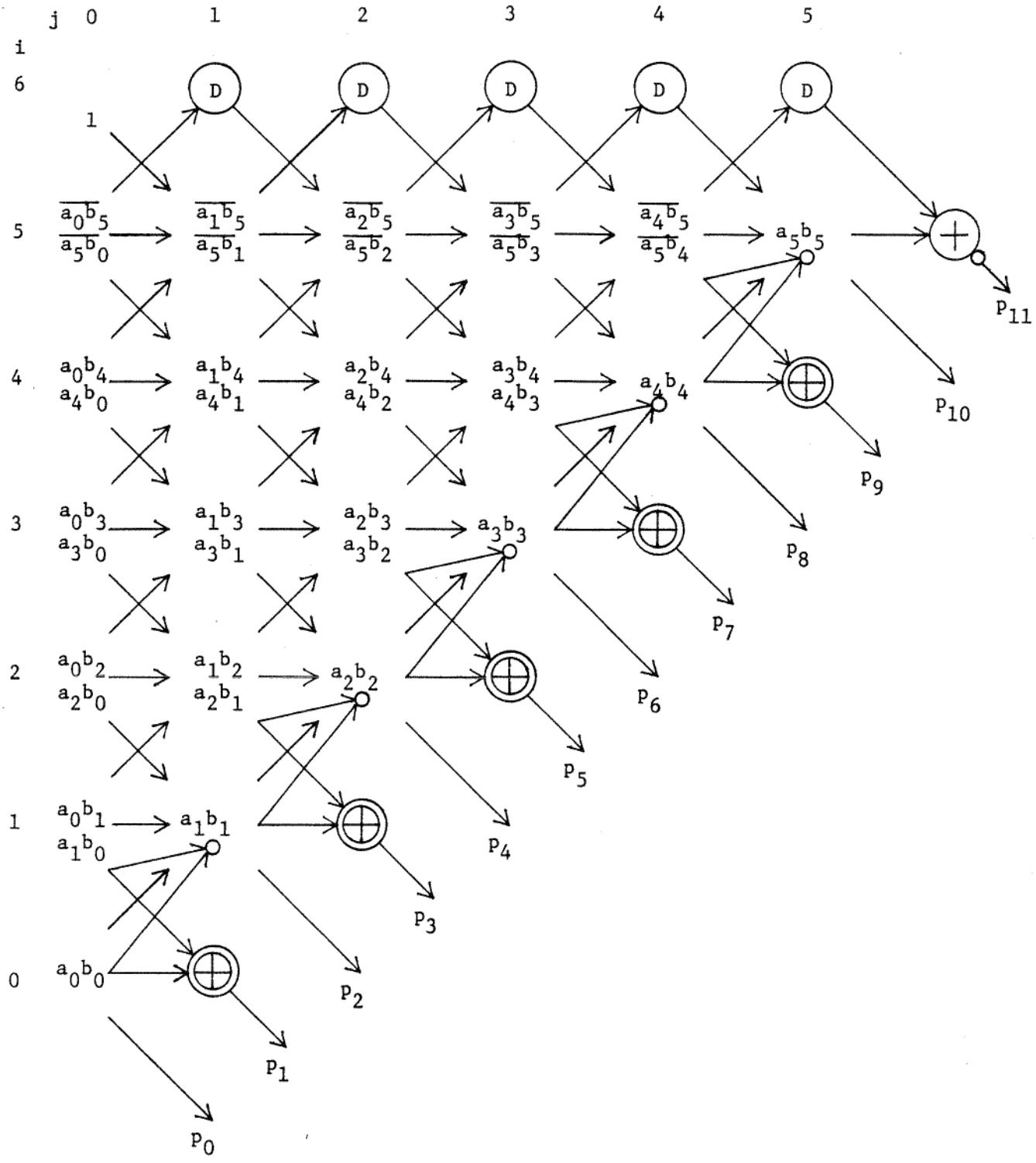


Figure 9. A 6 by 6 bit two's complement 5-counter multiplier.

6. Conclusion

We discussed iterative array algorithms for parallel multiplication which are suitable for VLSI single chip implementation. Of the four algorithms discussed we considered three methods for improving the multiplication speed of straightforward algorithms by a factor of two. We found that the radix 4 multiplier requires too much hardware, 16 times more than that of the binary full adder multiplier, while another algorithm, the (6,3,4) counter multiplier which is asymptotically as simple as the binary full adder multiplier, is feasible for only large n. We conclude that the iterative array algorithm based on 5-counter cells is a practical fast multiplication algorithm for VLSI single chip implementation.

References

[1] E.L. Braun, "Digital Computer Design," Academic Press, New York, 1963.

[2] C.S. Wallace, "A suggestion for a fast multiplier," IEEE Trans., Electron Comput. EC-13, Feb. 1964, pp. 114-117.

[3] L. Dadda, "On parallel digital multipliers," Alta Freq., No. 10, Vol. 45, 1976, pp. 574-580.

[4] W.J. Stenzel, W.J. Kubitz, and G.H. Garcia, "A compact high-speed parallel multiplication scheme," IEEE Trans. Comput., Vol. C-26, No. 10, Oct. 1977, pp. 948-957.

[5] S.D. Pezaris, "A 40-ns 17-Bit by 17-Bit Array Multiplier," IEEE Trans. Comput., Vol. C-20, No. 4, Apr. 1971, pp. 442-447.

[6] C.R. Baugh and B.A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. Comput., Vol. C-22, No. 12, Dec. 1973, pp. 1045-1059.

LIST OF CAPTIONS

Figure 1. (a) A 4 by 4 bit binary full adder multiplier.

(b) A full adder cell with a partial-product $a_i b_j$ as an input. The output value is calculated as: $2out_1+out_0 = in_0+in_1+a_i b_j$.

(c) A full adder cell without partial product. The output value is calculated as: $2out_1+out_0 = in_0+in_1+in_2$.

Figure 2. A 2 by 2 radix 4 multiplier.

Figure 3. The interconnection of radix 4 full adders. A circle indicates an adder of four radix 4 digits whose position in the array is shown by the indices in the circle.

Figure 4. (a) A radix 2 cell.

(b) A radix 4 cell.

Figure 5. A 5-counter cell. The output value is calculated as:

$$\sum_{i=0}^{2} out_i 2^i = a_i b_j + a_j b_i + \sum_{i=-2}^{0} in_i.$$

Figure 6. A 6 bit by 6 bit 5-counter multiplier. ⊕ indicates an exclusive OR and ⊕ indicates an inclusive OR.

Figure 7. A (6,3,4) counter cell. The output value is calculated as:

$$\sum_{i=0}^{3} out_i 2^1 = 2\left(a_j b_j + \sum_{i=0}^{4} in_{1i}\right) + \sum_{i=0}^{2} in_{0i}.$$

Figure 8. A 6 by 6 bit (6,3,4) counter multiplier.

Figure 9. A 6 by 6 bit two's complement 5-counter multiplier.

Index Terms: Parallel multiplication, iterative array, 5-counter, partial-products, complexity, multiplication speed

Footnotes

This work was supported in part by IBM Corporation, Essex Junction, VT, under an IBM Departmental Grant.

The author is with the Thayer School of Engineering, Dartmouth College, Hanover, NH 03755.