ON BIT SEQUENTIAL MULTIPLIERS

by RAVINDRA V. DONTHI, MOHAMMED SALEEM and HARPREET SINGH

Department of Electrical & Computer Engineering, Wayne State University, Detroit, U.S.A.

Abstract

Recently bit sequential multiplier algorithms have been found more useful in the area of interconnection of multiple processors within a VLSI structure [1],[2]. The object of the present paper is to suggest modified bit sequential algorithms to achieve more speed and to attain its conformity with other algorithms such as division, square-rooting etc. with a view to utilize them in future arithmatic arrays. In the present paper the following has been taken up: a) Bit sequential multiplier using carry look-ahead technique, b)Bit sequential multiplier using most significant bit first,and c) Nega-binary bit sequential multiplier

Introduction

Considerable interest has recently been shown in serial multiplication rather than parellel multiplication in the areas of interconnection of Multiprocessing, Arithmatic arrays involving division, squaring and square rooting etc. Design of fast and efficient multipliers is currently an active area of research. Many scientific and engineering applications such as Inversion of matrices, solution of differential equations etc. require large number of multiplications. The multipliers discussed in [1],[2] involve bit sequential input and output. The adders employed use the ripple carry methods for addition. The object of the present paper is to modify the existing bit sequential algorithms to achieve faster speed and hopefully conform to the other arithmatic operations such as division, square rooting etc. with a view to utilize them in future arrays. The paper has been divided into three parts. Part(a) discusses a Carry look-ahead bit sequential multiplier to achieve faster speed, Part(b) modifies the existing algorithm for taking most significant bit first for multiplication and Part(c) proposes an algorithm for multiplication of Nega-binary numbers and is suitable for VLSI implementation.

Part(a)

Bit sequential multiplier using Carry Look-ahead technique

Here we modify the C.B.S. multiplier [2] by using carry-look ahead adders.

The proposed algorithm will be as follows:

(1) Let $\underline{a}$ & $\underline{b}$ be the multiplier and multiplicand.

$$\underline{a} = \sum_{j=1}^{n} a_j . 2^{j-1} = [a_n, \dots\dots\dots\dots a_1]$$

$$\underline{b} = \sum_{k=1}^{n} b_k . 2^{k-1} = [b_n, \dots\dots\dots\dots b_1]$$

(2) The partial product P_i after the i^{th} iteration [2] becomes:

$$P_i = \sum_{j=1}^{i} a_j . 2^{j-1} . \sum_{k=1}^{i} b_k . 2^{k-1}$$

At $i=k$,

$$q_i = P_i * 2^{-1}$$

$$= [q_{i-1} + a_i(b_i,\dots,b_1) + b_i(a_i,\dots,a_1)]/2$$

(3) The three inputs for the i^{th} iteration and j^{th} module are given as (Table-1):

$$A(i,j) = \begin{vmatrix} b_i\ a_j & \text{for } j<1 \\ 0 & \text{otherwise} \end{vmatrix}$$

$$B(i,j) = \begin{vmatrix} a_i\ b_j & \text{for } j\leq 1 \\ 0 & \text{otherwise} \end{vmatrix}$$

$$S_j = A(i-1,j+1) \oplus B(i-1,j+1) \oplus S_{j+1} \oplus C_{j+1}$$

(4) The Logic expressions for output of the (3,2) counter modules are

$$L_j = A(i,j) \oplus B(i,j) \oplus S_j$$

$$G_j = [A(i,j).B(i,j)] + [A(i,j) \oplus B(i,j)].S_j$$

(5) The expression for the generation of the carry C_j is :

$$C_j=G_{j-1}+G_{j-2}[L_{j-1}]+G_{j-3}[L_{j-1},L_{j-2}]+\dots+G_1[L_{j-1}\dots L_2]$$

This multiplier requires only k number of iterations to obtain a complete product of 2k bits using k bit length operands and is faster than C.B.S. multiplier[2]. Further in this multiplier, at any given time during the multiplication the complete product of 2k bit length is available with respect to the k bits of multiplier and multiplicand received till that time.

The C.B.S. multiplier[2] employs ripple carry techniques. Here k number of excess iterations are spent just for the first and second order carries

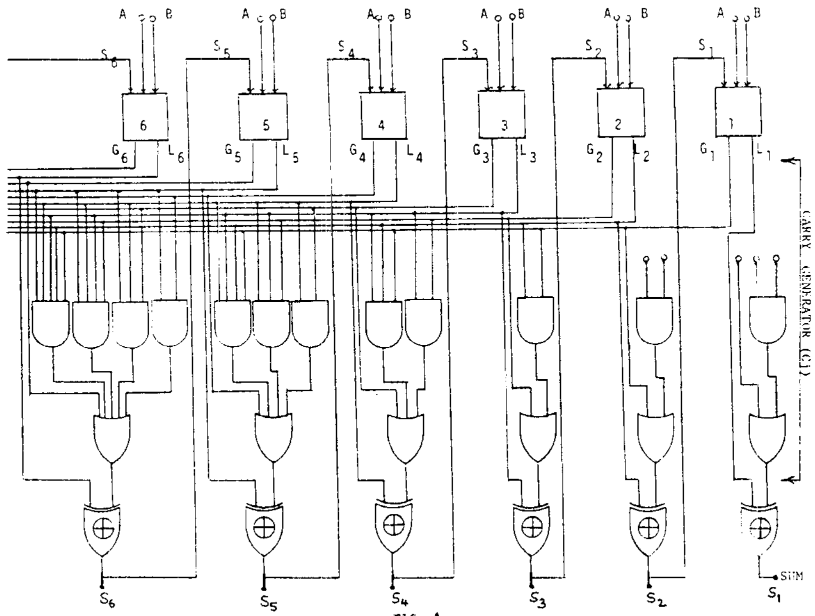


FIG. 1.

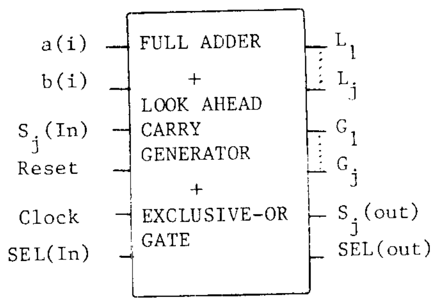


FIG. 2 CELL

~~to~~ ripple from least significant bits. Thus the precious time is saved in the present multiplier by generating the necessary carry in the same counter module. The number of counter modules required to perform multiplication for k bits operand is (k+1) modules. (Where k is the maximum number of bits in the multiplier or multiplicand). The construction of the multiplier is as shown in Fig.1. It consists of 6 counter modules suitable for multiplication of operands of n-bits length. However, the same set of modules can be extended to operands of n-bits length. Each counter module consists of a full adder, a carry generator and an 'Exclusive-OR' gate to form the sum bit.

The partial products are formed by adding together the previous partial products and two corrective terms A and B based on the currently known bits of a and b.

Table - 1.
A and B inputs for 5 bit operands.

Iteration ↓		Module(j) ← 6	5	4	3	2	1
1	A	0	0	0	0	0	0
	B	0	0	0	0	0	a_1b_1
2	A	0	0	0	0	0	b_2a_1
	B	0	0	0	0	a_2b_2	a_2b_1
3	A	0	0	0	0	b_3a_2	b_3a_1
	B	0	0	0	a_3b_3	a_3b_2	a_3b_1
4	A	0	0	0	b_4a_3	b_4a_2	b_4a_1
	B	0	0	a_4b_4	a_4b_3	a_4b_2	a_4b_1
5	A	0	0	b_5a_4	b_5a_3	b_5a_2	b_5a_1
	B	0	a_5b_5	a_5b_4	a_5b_3	a_5b_2	a_5b_1

As an illustration of the working of the bit sequential multiplier having carry look-ahead techniques, Table 1 and 2 give the step by step generation of 'A' and 'B' inputs and product during each iteration. As an example, the generation of carry generator is given below;

$$C_1 = 0 \quad , \quad C_2 = G_1 = 1 \quad , \; C_3 = G_2 + G_1L_2 = 1+0 = 1 \; ,$$

$$C_4 = G_3 + G_2L_3 + G_1L_2L_3 = 0+1+0 = 1$$

$$C_5 = G_4 + G_3L_4 + G_2L_3L_4 + G_1L_2L_3L_4 = 0+0+1+0 = 1$$

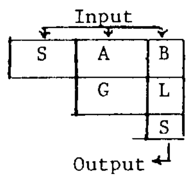
$$C_6 = G_5 + G_4L_5 + G_3L_4L_5 + G_2L_3L_4L_5 + G_1L_2L_3L_4L_5$$
$$= 0+0+0+1+0 = 1$$

Table - 2.

Inputs and output bits for 5 bit multiplicand and 5 bit multiplier are given in the format as shown:

Let a=10101
b=11011

a*b=1000110111



Iteration(i) ↓	Module(j) ← 6	5	4	3	2	1
1	000 00 0	000 00 0	000 00 0	000 00 0	000 00 0	001 01 1 → 1
2	000 00 0	000 00 0	000 00 0	000 00 0	000 00 0	010 01 1 → 1
3	000 00 0	000 00 0	000 00 0	000 00 0	001 01 1	001 01 1 → 1
4	000 00 0	000 00 0	000 00 0	010 01 1	000 00 1	110 10 0 → 0
5	000 00 1 ↓ 1	001 01 0 ↓ 0	001 01 0 ↓ 0	010 01 0 ↓ 0	101 10 1 ↓ 1	111 11 1 → 1

Part (b)

Bit sequential multiplier using most significant bit first

The multiplier algorithm proposed in [1],[2] utilise left shift in multiplication. However, right shift method of multiplication is preferred some times to left shift in division, squaring and square rooting algorithms. This right shift method has also been used by the various pipelined arithmatic arrays discussed recently [5]. In view of this it will be worthwhile extending the Bit sequential multiplication algorithms described in [1] to right shift algorithms. Here we propose a bit sequential multiplier algorithm using right shift multiplication. It may be noted that in such an algorithm we come across most significant bit first as compared to the other methods in which least significant bit is taken first.

The proposed algorithm, which is a modification of the algorithm by [1], will consist of the following:

The module used by [1] can be used for right shift multiplication (Fig.4a and b). However, the selection of modules for the five inputs will be given by the modified algorithm. The number of inputs to this module are 5 and 3 respectively as shown in Fig.4. The five inputs for the ith iteration and jth module are generated as follows;

$$A(i,j)=\begin{cases} b_{k-i+1}\cdot a_{j+i-k} & \text{for } 2(k-i)+1<j<2k-i \text{ and } i\leq k \\ 0 & \text{otherwise} \end{cases}$$

$$B(i,j)=\begin{cases} a_{k-i+1}\cdot b_{j+i-k} & \text{for } 2(k-i)<j<2k-i \text{ and } i\leq k \\ 0 & \text{otherwise} \end{cases}$$

$$C_1(i,j)=\begin{cases} 0 & \text{for } i=j=1 \\ S_{2,3}[A(i-1,j-1),B(i-1,j-1),C_1(i-1,j-1), C_2(i-1,j-1),S(i-1,j-1)]^1 \end{cases}$$

$$C_2(i,j)=\begin{cases} 0 & \text{for } i=j=1,2 \\ S_{4,5}[A(i-1,j-2),B(i-1,j-2),C_1(i-1,j-2), C_2(i-1,j-2),S(i-1,j-2)] \end{cases}$$

$$S(i,j)=\begin{cases} 0 & \text{for } i=1 \\ S_{1,3,5}[A(i-1,j),B(i-1,j),C_1(i-1,j), C_2(i-1,j),S(i-1,j)]^1 \end{cases}$$

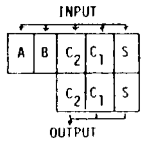
Here A and B are the modified inputs computed on the basis of $\underline{a}$ and $\underline{b}$ operand bits received for the ith module and are obtained as per the Table-3, where it is utilised for a 4 bit operand and can be extended for n bit operands.

'C_1' is the first order carry generated during (i-1)th iteration and is carried from (j-1)the module to jth module for the ith iteration. 'C_2' is the second order carry generated during (i-1)th iteration and is carried from (j-2)nd module to jth module for the ith iteration. 'S' is the sum bit computed during the (i-1)th iteration for the five inputs in the jth module and is preserved in the same module for computation during the ith iteration. For '$S_{1,3,5}[A,B,C_1,C_2,S]$', whenever the

Input and output bits for 4 bit multiplier and multiplicand are given in the Format as shown:

Let $\underline{a}$ = 1011
$\underline{b}$ = 1101

$\underline{a}$ X $\underline{b}$ = 10001111



ITERATION(i) ↓ MODULE (j) ←

ITERATION(i)	8	7	6	5	4	3	2	1
1.	00000	01000	00000	00000	00000	00000	00000	00000
	000	001	000	000	000	000	000	000
2.	00000	00001	10000	00000	00000	00000	00000	00000
	000	001	001	000	000	000	000	000
3.	00000	00001	00001	01000	01000	00000	00000	00000
	000	001	001	001	001	000	000	000
4.	00000	00001	00001	00001	11001	01000	10000	01000
	000	001	001	001	011	001	001	001
5.	00000	00001	00001	00011	00001	00001	00001	00001
	000	001	001	010	001	001	001	001
6.	00000	00001	00011	00000	00001	00001	00001	00001
	000	001	010	000	001	001	001	001
7.	00000	00011	00000	00000	00001	00001	00001	00001
	000	010	000	000	001	001	001	001
8.	00010	00000	00000	00000	00001	00001	00001	00001
	001	000	000	000	001	001	001	001
	↓	↓	↓	↓	↓	↓	↓	↓
	1	0	0	0	1	1	1	1

Fig.3

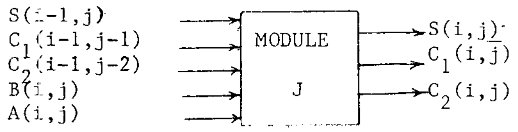


FIG. 4 (a) MODULE

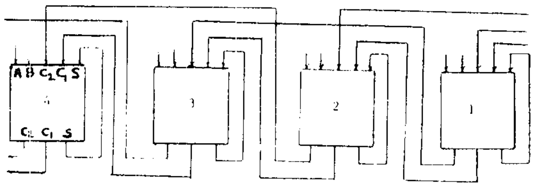


Fig. 4(b)

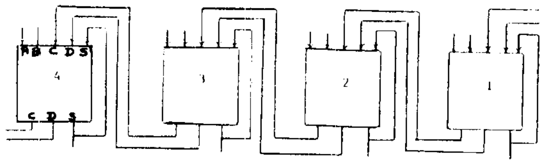


Fig. 5

Table - 3.

A & B inputs for 4 bit multiplicand & multiplier.

ITERATION (i) ↓		MODULE (j) ← 8	7	6	5	4	3	2	1
1.	A	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0
	B	0 0	a_4b_4	0 0	0 0	0 0	0 0	0 0	0 0
2.	A	0 0	0 0	b_3a_4	0 0	0 0	0 0	0 0	0 0
	B	0 0	0 0	a_3b_4	a_3b_3	0 0	0 0	0 0	0 0
3.	A	0 0	0 0	0 0	b_2a_4	b_2a_3	0 0	0 0	0 0
	B	0 0	0 0	0 0	a_2b_4	a_2b_3	a_2b_2	0 0	0 0
4.	A	0 0	0 0	0 0	0 0	b_1a_4	b_1a_3	b_1a_2	0 0
	B	0 0	0 0	0 0	0 0	a_1b_4	a_1b_3	a_1b_2	a_1b_1

sum of the arguments[A,B,C_1,C_2,S] equals 1 or 3 or 5, then the sum bit S will be a logic one otherwise will be a logic zero.

The number of counter modules required for right shift multiplication for a k bits operand is 2k and the number of iterations required to obtain the complete product will be less than or equal to 2k clock periods, assuming one clock period for each iteration. The only restriction to be observed in using this right shift multiplier is that both the operands a and b must be of equal bit length. for example a=0011 and b=1010. The 'A' and 'B' modified inputs are computed only for the first k number of iterations and for the next k iterations they assume logic zero's as given in Table-3. At the end of the 2k number of iterations the complete product will be available in the 'S' bits of the modules. An example as illustrated in Fig.3, gives the step by step computations for 4 bit operands.

Part (c)

Nega-binary bit sequential multiplier

Here we propose a Nega-binary bit sequential multipler. The nega-binary numbers have attracted the attention of several research workers because both negative and positive numbers can be represented without the use of seperate sign bit [10].

Let the $a_k \ldots\ldots a_1$ and $b_k \ldots\ldots\ldots b_1$ are the two k bit numbers. Their product is given by

$$\begin{aligned} P_{2k}\ldots\ldots P_1 &= [a_k \ldots\ldots\ldots a_1][b_k \ldots\ldots\ldots b_1] \\ &= [(-2)^{k-1}a_k + (a_{k-1}\ldots a_1)][b_k \ldots\ldots b_1] \\ &= [(-2)^{k-1}a_k(b_k \ldots\ldots b_1)] + \\ &\quad (a_{k-1}\ldots a_1)[(-2)^{k-1}b_k + (b_{k-1}\ldots\ldots b_1)] \\ &= (-2)^{k-1}a_k(b_k \ldots b_1) + (-2)^{k-1}b_k(a_{k-1}\ldots a_1) + \\ &\quad (a_{k-1}\ldots\ldots a_1)(b_{k-1}\ldots\ldots b_1) \end{aligned}$$

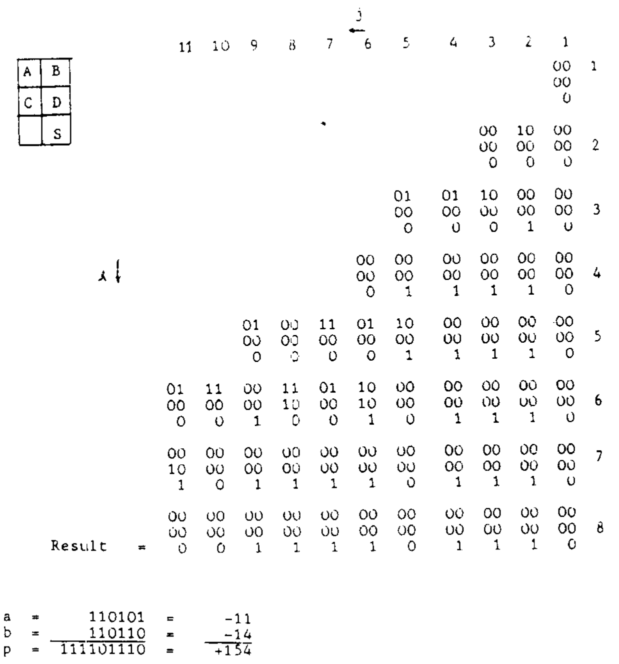


FIG. 6.

$$= (-2)^{k-1}[a_k(b_k \ldots\ldots b_1) + b_k(a_{k-1}\ldots\ldots a_1)] + P_{2k-2}\ldots\ldots P_1$$

The first term is the sum of products of the bits and the weight of this sum is $(-2)^{k-1}$.

For this multiplier, each module has five inputs and three outputs. The inputs are A,B,C,D and S where the outputs are S,C and D. C and D are the carries and the use of these carries can be fully understood by the table given below:

Input				Output		
C_i	D_i	A	B	S	C_{i+1}	D_{i+1}
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	1	0
0	1	1	1	1	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	1	0	0
1	1	0	0	x	x	x
1	1	0	1	x	x	x
1	1	1	0	x	x	x
1	1	1	1	x	x	x

The input equations for A,B,C and D are given as:

$$A(i,j) = \begin{cases} b_i a_{j-i+1} & \text{if } j < 2i-1 \\ 0 & \text{if } j \geq 2i-1 \end{cases}$$

$$B(i,j) = \begin{cases} a_i b_{j-i+1} & \text{if } j < 2i \\ 0 & \text{if } j \geq 2i \end{cases}$$

$$C(i,j)=\begin{cases} S_{2,3}[A(i-1,j-1),B(i-1,j-1), \\ \qquad S(i-1,j-1),D(i-1,j-1)] \\ S_4[A(i-1,j),B(i-1,j),D(i-1,j),S(i-1,j)] \end{cases}$$

$$D(i,j)=\begin{cases} 1 \text{ if } A(i-1,j-1)=B(i-1,j-1)=S(i-1,j-1)=0 \ \&C=1 \\ S_4[A(i-1,j),B(i-1,j),D(i-1,j),S(i-1,j)] \\ 1 \text{ if } C(i-1,j-1)=1 \end{cases}$$

The interconnected modules are as shown in Fig. 5 and the example describing multiplication is given in Fig.6.

It may be noted that Nega-binary adders require two carries as compared to one carry required in binary adders. The binary multipliers proposed by Chen and Willoner and Strader & Rhyne require two carries C_1 and C_2 for bit sequential multiplier. The negabinary multiplier proposed here performs multiplication by generating only two carries C_i & D_i. Normally, one may think that a negabinary multiplier should generate four carries. However in the negabinary multiplier used here, the object is achieved by using only two carries and not more. This development is based on utilising the carries C_i and D_i. In normal addition both cannot be 1 at any one time. So whenever an extra carry (as C_2 in Chen Willoner multiplier) is needed, both C_i and D_i are made 1.

Conclusion

The algorithm given by [2] has been modified by using carry look-ahead adders instead of ripple carry adders to achieve faster speed and to conform them to other operations such as division, square rooting etc. A cell suitable for VLSI implementation is also given in Fig.2. Neddless to say the cell will be more complex as more logic gates will be required. However, the proposed procedure is faster as the number of iterations required by the algorithm is exactly only half of the iterations required in C.B.S. algorithm.

A bit sequential algorithm using right shift method and by taking the most significant bit first is also proposed. The proposed algorithm is the modification of the algorithm proposed by [1],[2]. Such an algorithm will be more useful in future arithmatic arrays in view of the conformity of right shift multiplication with the division and square rooting algorithms. It may be noted that in the proposed algorithm only the input bits are given in the sequential order, while the output bits will be obtained in parallel (Fig.3). This may appear to be a disadvantage. However it is hoped that such an algorithm will result in a simpler overall VLSI structure in the implementation of Arithmatic arrays involving division, squaring and square-rooting etc.

Finally an algorithm for the multiplication of negabinary numbers is proposed. The proposed multiplier is a bit sequential one and is also suitable for VLSI implementation.

References

[1] I-ngo Chen and Robert Willowner, An O(n) parallel multiplier with bit sequential input and output, IEEE Transactions on Computers,oct. 1979.

[2] Noel R. Strader and V. Thomas Rhyne, A canonical bit sequential multiplier, IEEE Transactions on Computers, Aug.1982.

[3] K.Hwang, 'Computer Arithmatic', John Wiley, 1979, P 387.

[4] Peter M. Kogge, 'Architecture of pipeline computers', Macgraw Hill book co. 1981.

[5] A.K.Kamal, Harpreet Singh and D.P.Agarwal, 'A Generalized pipeline array', IEEE Transactions on Computers, May, 1974.

[6] J.C.Majithia and R.Kitai, 'An arry for multiplication of signed binary numbers.' IEEE Transactions on Computers, Vol C-20,pp 214-216 Feb. 1971.

[7] S.Singh and R.Waxman, 'Multiple operand addition and multiplication', IEEE Transactions on Computers, Vol C-22, pp 113-120, Feb.1973.

[8] S.Bandyopadhyay, S.Basu and A.K.Choudhary, 'An iterative array for multiplication of signed binary numbers', IEEE Transactions on Computers, Vol C-21, pp 921-922, Aug.1972.

[9] A.Habibi and P.A.Wintz, 'Fast Multipliers', IEEE Transactions on Computers,Vol C-19, pp 153-157, Feb. 1970.

[10] G.F.Songster, 'Negative base number representation systems', IEEE Transactions on Computers, Vol EC-12, pp 274-277, June 1963.

- o -