International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 10 (2019), pp. 1757-1763
© International Research Publication House. http://www.irphouse.com

Design of A Fast Multiplier with (*m*, 3)-Adders

Myungchul Yoon

Professor, Department of Electronics and Electrical Engineering,
Dankook University, Republic of Korea.

ORCID: 0000-0001-7952-4349

Abstract

The (7, 3)-adder based multiplier design is presented in this paper. A (*m*, 3)-adder ($4 \le m \le 7$) accumulates *m* bits at a time and produce 3 output: S, C1, and C2 such that (C2 C1 S) represents the number of 1s in the m-bit. The (7, 3)-adder based multiplier uses a (7, 3)-adder as the basic unit and other (*m*, 3)-adders as auxiliary units. An algorithm to build a (7, 3)-adder based tree, called Y-tree, is presented. The height of Y-tree is about half that of the Wallace-tree so that a Y-tree multiplier could be faster than a Wallace-tree multiplier depending on the speed of (7, 3)-adder. By focusing on delay minimization, (*m*, 3)-adder circuits are designed and it is verified that the (7, 3)-adder based multipliers can be faster than (3, 2)-adder based multipliers.

Keywords: High-Speed multiplier design, Multiplier design, Wallace-tree multiplier, (m, 3)-adder design.

I. INTRODUCTION

The multiplier is an essential element for microprocessors, digital signal processors and graphic processors so that it is one of the key hardware blocks in most digital signal processing systems. Therefore, various studies have been conducted to produce a high speed multiplier [1]-[6].

The most basic form of multiplication consists of forming the product of two unsigned binary numbers. This can be accomplished through the traditional technique taught in primary school, simplified to base 2. The process to obtain the final result is composed of additions of partial products. The speed of a multiplier is determined by the method of manipulating the partial products.

The array multiplier [1] adds partial products sequentially with carry save addition (CSA) scheme. The Wallace multiplier [2] uses parallelism in accumulating all bits in a digit. The Booth encoded multiplier [4][5] employs an encoding algorithm which can reduce the number of partial products.

For a 64-bit multiplier, 64-partial products (or 32 for Booth multipliers) are produced, and these partial products are added by the digit-wise addition for parallelism like Wallace multipliers. The number of bits in a digit ranges from 1 to 64, and most of digits have tens of bits to be added. Most multipliers use a (3, 2)-adder as the basic unit. A (3, 2)-adder, called full-adder in general, adds 3 bits at a time and produce a sum (S) and a carry (C), so that 3 bits are removed and 2 bits are newly included in a digit. Therefore, one parallel addition using (3, 2)-adders reduces about 1/3 of the bits in a digit. If there is an adder which can reduces more bits than a (3, 2)-adder, it may afford to increase the speed of multipliers.

The (7, 3)-adder based multiplier design is proposed in this paper. An (*m*, 3)-adder ($4 \le m \le 7$) adds *m* bits at a time and produces 3 outputs, S (sum), C1 (carry1), and C2 (carry2), such that the binary number made by (C2 C1 S) represents the number of 1s in the *m* bits. The (7, 3)-adder based multiplier uses a (7, 3)-adder as the basic unit and other (*m*, 3)-adders as auxiliary units. A (7, 3)-adder removes 7 bits but produces 3 new bits so that parallel additions using (7, 3)-adders can reduce 4/7 of bits in a stage. Therefore, the (7, 3)-adder can add all bits in a digit with fewer stages than the (3, 2)-adder.

The disadvantage of (*m*, 3)-adders is the complexity of their circuits. The (*m*, 3)-adder circuits are more complex than the (3, 2)-adder circuit so that they are slower, larger, and consumes more power than the (3, 2)-adder. However, the (7, 3)-adder based multiplier requires much fewer adders and stages than the (3, 2)-adder based multiplier. Therefore, the former could yeild better performance than the latter depending on the design of (*m*, 3)-adder circuits.

A design of high-speed multiplier with (*m*, 3)-adders is presented in this paper. The advantages of a (7, 3)-adder based multiplier design are described in Section II. The algorithm for building a tree with (*m*, 3)-adders is presented in Section III, and the (*m*, 3)-adder circuits designed for high speed applications are presented in Section IV. The speed of these circuits are evaluated in Section V, and Section VI concludes this paper.

II. (7, 3)-ADDER BASED MULTIPLIER DESIGN

Multiplication of two *N*-bit numbers $P=YX$ can be viewed as forming *N* partial products of *N* bits each, and then summing the appropriately shifted partial products to produce an 2*N*-bit result *P*. For example, the multiplication of two 6-bit binary numbers is performed as in Fig. 1. The p_i is obtained by adding all bits in the column and the carries produced from p_{i-1} column. For a 64-bit multiplier, hundreds of bits should be added to obtain a p_i.

Those bits are added by many layers (or stages) of adder. All adders that belong to a layer can operate simultaneously, but

International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 10 (2019), pp. 1757-1763
© International Research Publication House. http://www.irphouse.com

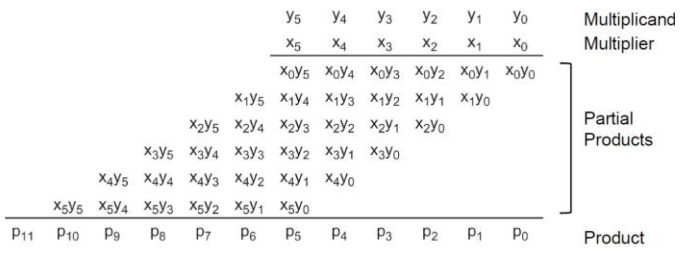
Multiplicand
Multiplier
Partial Products
Product

Fig. 1. Partial products

adders in different layers must operate in different time. Adders in a layer require the results of adders in the lower layer so that each layer should operate sequentially. A kind of carry propagation adder (CPA) of which inputs are the outputs of the last layer is necessary to obtain the final result *P*.

An N-bit array multiplier adds the bits in a digit serially by N layers. Although a regular structure is its an advantage, many layers increase the delay especially for a large N. Parallelism is employed to reduce the number of layers. The Wallace-tree multiplier adds all bits in a digit in parallel with multiple adders as in Fig. 2. Fig. 2 shows the addition process to obtain p_{15} for a 16-bit Wallace-tree multiplier. It needs only 6 layers and a CPA while a 16-bit array multiplier requires 15 layers and a CPA. The number of layers is often called as the height of the tree. Since the layers operate sequentially, the height of tree is an important factor affecting the delay of multipliers.

Most of multipliers use (3, 2)-adder as the basic adder and (2, 2)-adder as the auxiliary adder. The (3, 2)-adder which is called full adder conventionally adds 3 bits of the same weight and produce 2 outputs: S (sum) and C (carry). While S has the same weight with the input, the weight of C has 2 times greater than that of inputs. The (2, 2)-adder, usually called half-adder, produce S and C as well, but it can add only two bits of the same weight. For a 64-bit multiplier, 64 partial products are produced so that maximum 64 bits in a digit should be added to obtain the results. In the Wallace-tree multiplier, those bits are added by many (3, 2)-adders in parallel. Each (3, 2)-adder takes 3 bits in a same digit and then produces S and C. S is left in the same digit while C is passed to the next digit. As the result, 3 bits per adder are removed from a digit but two new bits (one bit from the adder, and one bit from the lower digit) enter to the digit. Therefore, about 1/3 of bits in a digit are reduced by a (3, 2)-adder layer. Due to this reduction ratio of the full-adder, the height of 64-bit Wallace-tree multiplier becomes 10. Although it is much smaller than the height of array multipliers, it is still high. To increase the speed of a multiplier, it is necessary to reduce the height of tree.

Two approaches are possible for lowering the height of a tree. One is reducing the number of partial products like the Booth encoded multiplier. Radix-4 Booth encoded multiplier can reduce a half of partial products, but the height of tree decreases only 1. The other is increasing the reduction ratio of a layer which is effective in reducing the height of tree.

(7, 3)-adder based multiplier intended to make a fast multiplier circuit by lowering the height of tree. An (*m*, 3)-adder ($4 \le m \le 7$) adds *m* bits of the same weight (digit) and produces 3 outputs:

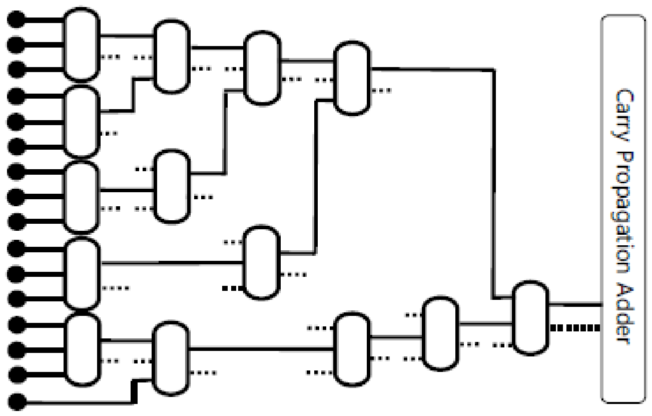

Fig. 2. Addition process of a 16-bit Wallace-tree multiplier

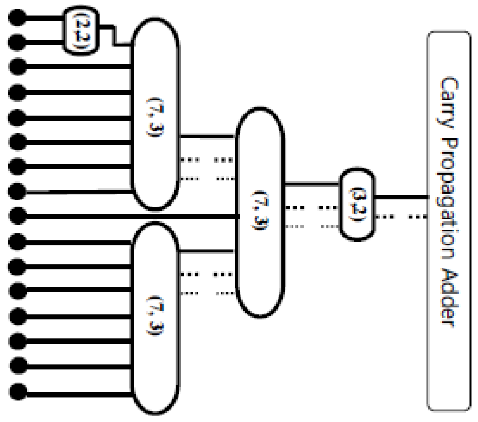

Fig. 3. Addition process of a 16-bit (7, 3)-adder based multiplier

S (sum), C1 (carry-1), and C2 (carry-2). S has the same weight with the inputs while the weights of C1 and C2 are 2 and 4 times greater than that of inputs respectively. Each adder in a (7, 3)-adder layer removes 7 bits in a digit but 3 new bits enter the digit so that 4/7 of bits in a digit are removed by a layer. Therefore, the reduction ratio of (7, 3)-adder is almost double that of (3, 2) –adder.

The (7, 3)-adder based multiplier design uses a (7, 3)-adder as the basic unit and other (*m*, 3)-adders, (3, 2)-adder and (2, 2)-adder, as auxiliary units. Fig. 3 shows the addition process for a 16-bit (7, 3)-adder based multiplier. Generally, the last layer is composed of (3, 2)-adders because less than or equal to 3 bits are left in each digit. The first layer of (2, 2)-adder is used to trim the number of bits as 2^k-1. A separate layer is necessary for trimming up to 16-bit multipliers, but adders for the trimming can be merged to the first (7, 3)-adder layer so that it can be removed from 32-bit multiplier (see the next section). Compared to Fig. 2, a 16-bit Wallace-tree multiplier has 6 while the new design has 4 layers.

Assuming that the last (3, 2)-layer and the CPA of both multipliers have the same delays, let us except them in comparison. Then, the number of (7, 3)-adder layers is less than a half of that of the Wallace-tree multiplier. In general, if the number of partial products is 2^k, the (7, 3)-adder based

```
// The adder-allocation algorithm for Y-tree multiplier
// This algorithm is valid for $N\times N$ ($N=2^m$) multiplier
// with the constraint that the maximum number of bits
// to be added in a digit (Pmax) is $Pmax < 2^h+4$ ($h \le m$)

ADDER_ASSIGNEMENT (N, p[.] )
  p[j] ;             // The number of bits in the j-th digit   ($0\le j\le 2N-1$)
  adder[i][j][k] ;   // the number of k-input adders
                     // arranged to the j-th digit in the layer i
  c[j] ;             // the number of carry-bits enterring
                     // to the j-th digit in the next layer

Begin
Clear_All ( adder[.][.][.] ) ;
Clear_All ( c[.] ) ;
Load (p[.] )
bottom_layer = 0 ;
Pmax= MAX(p[0], …, p[2n-1]) ;

// Trimming step - the adders in this layer can be operated
// in parallel with the next layer if h >4

Find_Integer (h)  // h satisfying  $Pmax < 2^h + 4$ and $h \le m$
if ( $Pmax > 2^h -1$ )
    for (j=0; j < 2N  ; j++) {   // assign adders for the j-th digit
        T = p[j] + c[j] - ( $2^h$ -1)
        if (T > 0 ) {
            adder[0][j][ T+1 ] = 1 ;
            p[j] = p[j] - T + 1 ;
            c[j+1] = c[j+1] + 1 ;
            if( T > 2 )   c[j+2] = c[j+2] + 1 ;
            }
        p[j] = p[j] + c[j] ;
        if (h < 5 ) bottom_layer = 1 ;
       }
// End of trimming

// beginning of adder- allocation for each layer
for (i = bottom_layer ;  ; i++) {
   Pmax= MAX(p[0], …, p[2n-1]) ;
   Clear_All ( c[.] ) ;
   if (Pmax == 2 ) { arrange a Carry Propagation Adder; Exit ;}
    else if (Pmax == 3 )  {
      for (j=0; j < 2N  ; j++) {
         if (p[j] + c[j] ==3) {
             adder [i][j][p[i]] = adder [i][j][p[i]] + 1 ;
             c[j+1] = 1 ;
             p[j] = 1  ; }
         p[j]= p[j] + c[j] ;  }
    else if (Pmax < 8) {
      for (j=0; j < 2N  ; j++)  {
         if (p[j] + c[j] > 3) {
             adder [i][j][p[i]] = adder [i][j][p[i]] + 1 ;
             c[j+1] = c[j+1] +1 ;
             if( p[j] > 3) c[j+2] = c[j+2] +1 ;
             p[j]= 1 ;  }
          p[j]= p[j] + c[j] ; }
   else {
     for (j=0 ; j < 2N  ; j++)  {
        L = $\lfloor p[j] / 8 \rfloor$ ; R = p[j] - 8 × L :
        adder [i][j][7] = L  ;
        c[j+1] = c[j+1] + L ;
        c[j+2] = c[j+2] + L ;
        p[j] = 2 × L ;
        if (R + (c[j] -2×L) > 3) {
             adder [i][j][R] = adder [i][j][R] + 1 ;
             c[j+1] = c[j+1] +1 ;
             if( R > 3) c[j+2] = c[j+2] +1 ;
             p[j] = p[j] +1 ; }
       p[j]= p[j] + c[j] ;  }
     }
End
```

Fig. 4. The pseudo-algorithms for building Y-tree

multiplier consists of k-2 of (7, 3)-layers and one (3, 2)-layer while the Wallace-tree multiplier has 2(k-1) of (3,2)-layers. Therefore, the height of (7, 3)-adder based multiplier is half that of the Wallace-tree multiplier.

III. An ADDER ARRANGEMENT ALGORITHM FOR THE (7, 3)-ADDER BASED MULTIPLIER DESIGN

For the (7, 3)-adder based multiplier design, 5 other adders, (2, 2), (3, 2), (4, 3) (5, 3), and (6, 3) adder can be used as auxiliary units. Since the number of bits to be added in a digit is different digit by digit, the type and the number of adders required to add these bits must be different digit by digit. Therefore, an algorithm is necessary that decides the efficient number and the type of adders for a given number of bits. A pseudo-algorithm for adder-allocation is suggested as in Fig. 4. Based on the number of bits in a digit, the algorithm determines the type and the number of adders for each layer of the tree.

The algorithm presented in Fig. 4 is valid only for design of $N\times N$ multiplier with $N = 2^m$ bits. For the multiplication of two N -bit binary numbers, N partial products are produced as in Fig. 1. The number of bits to be added in a digit varies from 1 to N along the position of a digit. Let p_i represent the number of bits to be added in $\underline{i}$-th digit. The maximum value of p_i ($p_{i\max}$) is N for usual multiplier design. Many multipliers employ Booth encoding algorithm to reduce the number of partial products. For radix-2^r Booth-encoded multipliers, the number of partial products can be reduced to $N/r \pm 1$.

For any design, the adder allocation algorithm in Fig. 4 can be applied when the number of partial products is less than $2^k + 4$. The first step of the algorithm is trimming $p_{i\max}$ to $2^k - 1$ with

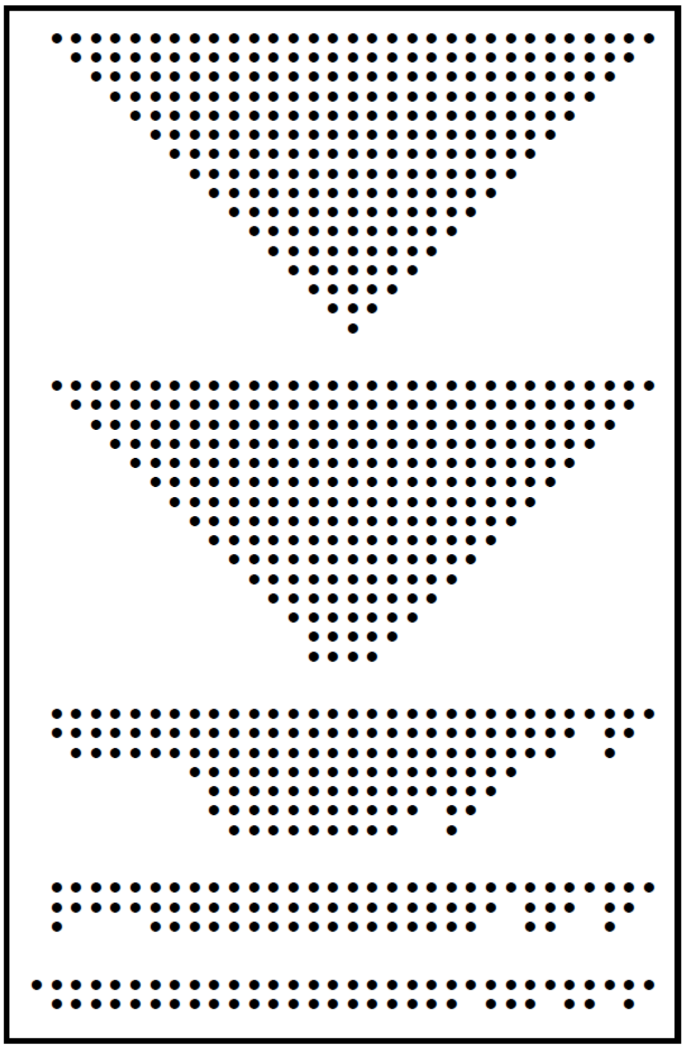

Fig. 5. Dot-diagram of a 16-bit Y-tree multiplier

several adders when the number of partial products is greater than or equal to 2^k. If p_{imax} becomes 2^k-1, it is decreased to $2^{k-1}-1$ by 2^{k-3} of (7, 3)-adders**.** In this way, p_{imax} can be reduced down to 3 by using k-2 layers of (7, 3)-adder. If p_{imax} reaches 3, no more (*m*, 3)-adders are needed, so that it is terminated by a (3,2)-adder layer followed by a kind of CPA.

If k > 4, the trimming steps can be performed simultaneously with the first (7, 3)-adder layer, so that it does not need count the trimming step as a separate layer. However, it is impossible to merge trimming step into another layer if k ≤ 4 so that it should be counted as a separate layer as in Fig. 3.

The tree generation by the algorithm is called Y-tree to distinguish it from the Wallace-tree which is based on (3, 2)-adder. Fig. 5 is the dot diagram for a 16-bit unsigned integer Y-tree multiplier which shows how the number of bits changes by operation of the layers.

IV. (*m*, 3)-ADDER CIRCUIT DESIGN

The height of Y-tree multiplier is less than half that of Wallace-tree multiplier. However, the delay per layer is longer since the

Table 1. Comparison of the Wallace-tree and the Y-tree for unsigned number multiplier.

		16-bit		32-bit	
		W-Tree	Y-Tree	W-Tree	Y-Tree
# of adders	(2, 2)	34	7	51	15
	(3 ,2)	188	30	909	46
	(4, 3)	-	6	-	13
	(5 ,3)	-	6	-	29
	(6, 3)	-	6	-	40
	(7, 3)	-	33	-	164
	CPA	1	1	1	1
# of layers	(2,2)		1		0
	(7, 3)	-	2		3
	(3 ,2)	6	1	8	1
	CPA	1	1	1	1

delay of a (7, 3)-adder is longer than a full-adder. Generally, delay, size and power of an adder increase rapidly with the number of inputs. Therefore, the design of fast and simple (*m*, 3)-adder circuits is the most important requirement for Y-tree multipliers.

Some comparisons between the Wallace-tree multiplier and the Y-tree multiplier for unsigned numbers are presented in Table 1. Inducing from the Table 1 to a 2^k bit multiplier, the height of Wallace-tree multiplier is 2(k-1) except a CPA, and that of Y-tree is k-1 if k > 4. Among the k-1 layers of Y-tree multiplier, one layer is always (3,2)-adder layer. If t_3, t_7, and t_{cpa} is the delay of (3, 2)-adder, (7, 3)-adder, and CPA respectively, the delay of the Wallace tree can be roughly estimated by

$$T_W = 2(k-1)t_3 + t_{cpa,W} \qquad (1)$$

and the delay of the Y-tree multiplier is

$$T_Y = (k-2)t_7 + t_3 + t_{cpa,Y} \qquad (2)$$

By assuming that the delay of CPAs is the same ($t_{cpa,W} = t_{cpa.Y}$), the Y-tree multiplier has the speed benefit against Wallace–tree multiplier when

$$t_7 \le (2 + \frac{1}{k-2})t_3 \qquad (3)$$

Therefore, Eq. 3 could be the speed limit of (m, 3)-adder3 circuit design.

Other important features for circuit design are power and size of the circuits. As in Table 1, the number of (7, 3)-adders to build a Y-tree multiplier is about 1/5~1/4 of the number of (3, 2)-adders required to build a Wallace-tree multiplier. The speed and area overheads of (4, 3) and (5, 3)-adder are relatively small compared to (6, 3) and (7, 3)-adder. Therefore, if we count the number of (6, 3) and (7, 3)-adders only, the following ratio is obtained

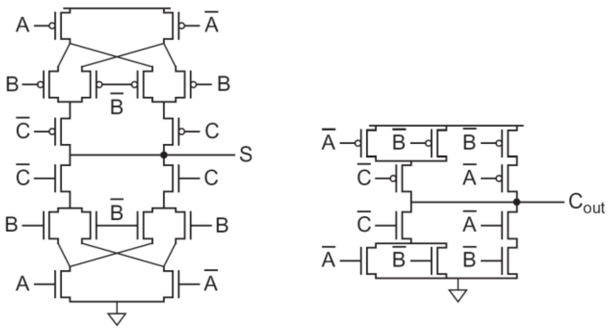

Fig. 6. A circuit of t (3, 2)-adder (full-adder)

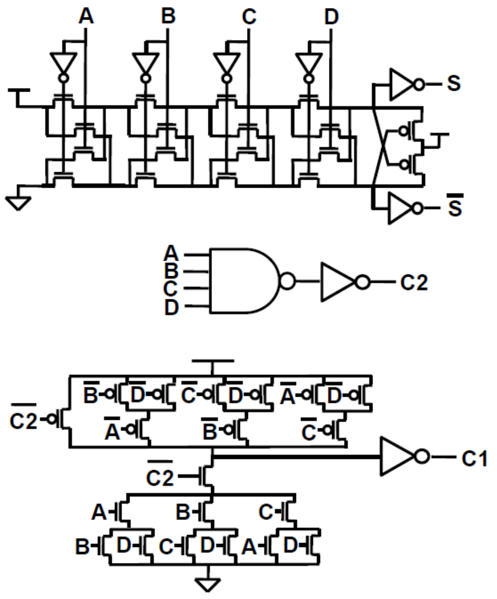

Fig. 7. The (4, 3)-adder circuit design

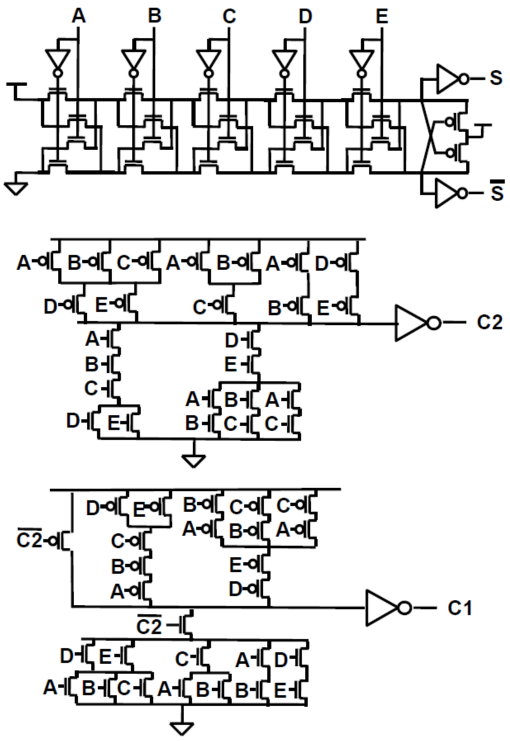

Fig. 8. The (5, 3)-adder circuit design

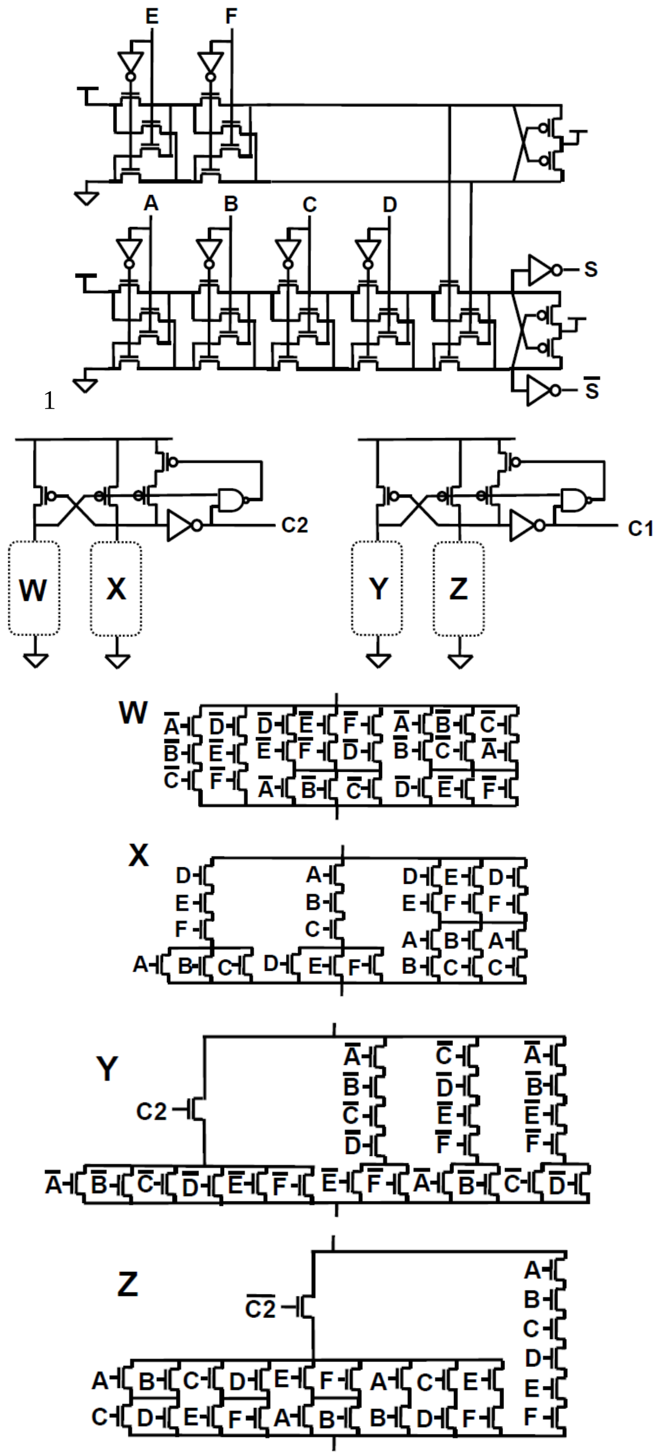

Fig. 9. The (6, 3)-adder circuit design

$$\frac{N_{3,W} - N_{3,Y}}{N_{7,Y} + N_{6,Y}} \approx 4.0 \sim 4.2$$

From this ratio, we can roughly obtain the power and size limits for (7, 3)-adder circuit design.

$$P_7 \le 4P_3 \qquad (4)$$

$$\text{Area}_7 \le 4\text{Area}_3 \qquad (5)$$

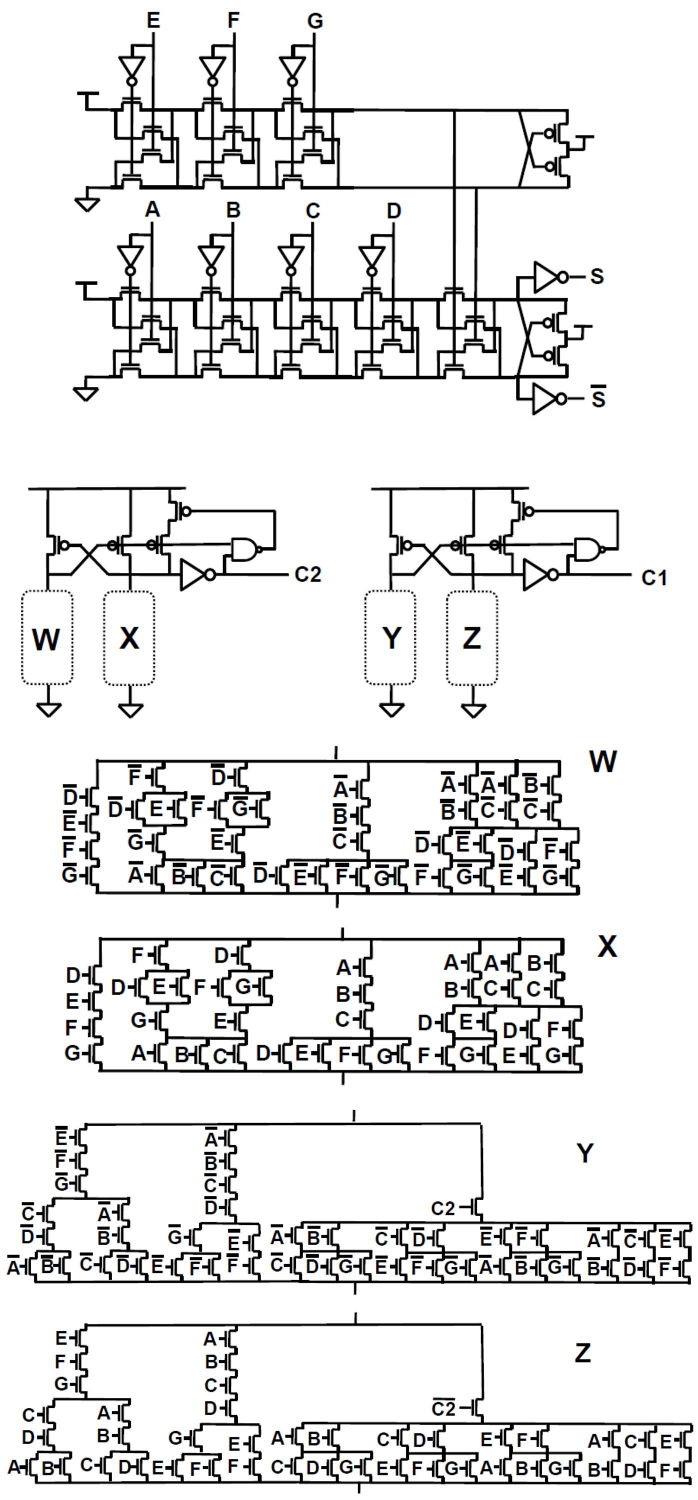

Fig. 10. The (7, 3)-adder circuit design

The Eq. 3, 4, and 5 are the conditions that the Y-tree multiplier can be a more efficient design than the Wallace-tree multiplier.

Various types of (*m*, 3)-adder circuit have been devised and their speeds, powers and sizes checked, but it is found that the design of a (7, 3)-adder meeting the Eq. 4 and 5 is a very challenging problem so far. It is easy to design (4, 3) and (5, 3)-adder which satisfy Eq. 3, 4, and 5, but it could not find

Table 2. The worst case delay of (*m*, 3)-adders

	Ref.	(*m*, 3)-adder			
	(3, 2)	(4, 3)	(5, 3)	(6, 3)	(7, 3)
Delay, t_m (ps)	280	315	381	422	528
t_m/t_{ref}	1.00	1.13	1.36	1.51	1.89

a (7, 3)-adder satisfying Eq, 4 and 5 so far. However, it is possible to design a (7, 3)-adder which satisfies Eq. 3. Until now, therefore, the Y-tree multiplier is a multiplier suitable for speed oriented applications which can take power and area overheads.

The circuit of (*m*, 3)-adder is depicted in Fig. 7~Fig. 10. Fig. 6 is the circuit diagram of the full-adder [7] used as the reference (3, 2)-adder.

The (4, 3)-adder and (5, 3)-adder circuits can be designed easily to satisfy Eq, 3, 4, and 5, but design of an efficient (6, 3)-adder and a (7, 3)-adder is very difficult. Since the complexity of an adder circuit is exponentially increase with the number of inputs, the design of (*m*, 3)-adder is focused on the (7, 3)-adder circuit design.

V. SPEED EVALUATION

The speed of newly designed adder is estimated immediately to check whether it satisfies Eq. 3. Delay simulations are performed by HSPICE with IBM's "1.2V-0.13μm 8RF-LM" model parameter.

The worst case delay of newly designed (*m*, 3)-adder circuits is listed in Table 2. The delay of an adder is measured by cascading ten identical adders (fanout=1). The delays are measured for various combination of input changes and the largest delay is selected.

Generally in the circuits in Fig. 7~10, the worst case delay of S-circuit is smaller than that of C1 and C2 circuits for those designs. Because C2 is used as one of inputs for C1-circuit, the C1-circuit is slower than the C2-circuit so that the worst case delay of the (*m*, 3)-adders occurs from the C1-circuits.

The design of the (4, 3)-adder and the (5, 3)-adder is relatively easy because their speed overhead is small. As in Table 2, the delays of a (4, 3)-adder and a (5, 3)-adder are about 13% and 36% longer than that of the full-adder. The (7, 3)-adder is designed by modifying the CVSL logic. For a conventional CVSL-type design, there is a big mismatch between the rising delay and the falling delay of the circuit. By the modification like in Fig. 10, the mismatch can be reduced. Its delay is about 1.89 times longer than that of the full-adder which satisfies Eq. 3.

It shows that the design of a (7, 3)-adder which satisfies the Eq. 3 is possible so that the (7, 3)-adder based multiplier can operate faster than the full-adder based multiplier.

V. CONCLUSION

A new method for multiplier design is suggested in this paper. Unlike previously developed multipliers which use a full-adder as the basic building unit, the newly suggested multiplier uses a (7, 3)-adder as the basic unit, with other (*m*, 3)-adders (*m* =4, 5, 6) as auxiliary units.

Similarly to the Wallace-tree of full-adders, a tree of (*m*, 3)-adders, called Y-tree can be used to make a multiplier. An algorithm making the Y-tree is presented. The (7, 3)-adder based multiplier design requires much fewer number of adders and lower height of tree. The height of Y-tree is about half that of the Wallace-tree so that a Y-tree multiplier can be faster than its Wallace-tree counterpart if the delay of (7, 3)-adder is less than 2 times of the delay of the full-adder. A set of (*m*, 3)-adder circuits are designed and verified by simulations such that their speeds satisfy the condition.

Unfortunately, the newly designed (7, 3)-adder circuit is much bulkier than the full-adder so that the Y-tree multiplier has disadvantages in size and power. So far, the (7, 3)-adder based multiplier is recommended only for speed-oriented applications. However, the limitation could be removed by developing an efficient (7, 3)-adder circuit. Out next research goal is to develop the design of a smaller and faster (7, 3)-adder circuit.

REFERENCES

[1] M. C. Park, B. W. Lee, G. M. Kim, and D. H. Kim, 1993, “Compact and Fast Multiplier Using Dual Array Tree Structure”, *IEEE Int. Symp. on Circuit and Systems,* Chigago, vol. 3, pp. 1817-1820.

[2] C. S. Wallace, 1964, “A Suggestion for a Fast Multiplier”, *IEEE Trans. Electron. Compt.*, vol. 13, no. 1, pp. 14-17.

[3] L. Dadda, 1965. Some Schemes for Parallel Multipliers, Alta Frequenza.

[4] A. D. Booth, 1951, “A Signed Binary Multiplication Technique,” *Jour. of Mech. Appl. Math.*, vol. 4, pp. 236-240.

[5] W. C. Yeh and C. W. Jen, 2000, “High-Speed Booth Encoded Parallel Multiplier Design*”,IEEE Trans. on Compt*., vol. 49, no. 7, pp. 692-701.

[6] P. Mehta, D. Gawali, 2009, “Conventional versus Vedic mathematical method for Hardware implementation of a multiplier”, *IEEE Int. Conf. on Advances in Computing, Control, and Telecommun. Technologies,* pp. 640-642.

[7] Neil H.E. Weste, and David M. Harris, 2010, Kamran, Principles of CMOS VLSI Design: A systems perspective, Pearson, Fourth Edition.