Improving the Recursive Multiplier

John Kim
Embedded Platform Solutions - Motorola, Inc
7700 Parmer Lane, Building C, PL31
Austin, TX 78729

Earl E. Swartzlander, Jr.
Electrical and Computer Engineering Department
University of Texas at Austin
Austin, TX 78712

Abstract

This paper examines the recursive multiplier and some potential enhancements for it. The delay of the recursive multiplier is similar to Dadda/Wallace fast multipliers, but the complexity is higher. Since the increase in complexity comes from the reduction stages, modified reduction cells are defined and compared to the existing design. The modifications take advantage of the fact that some of the inputs in the reduction cells are not used. Because of the simplicity of array multipliers in comparing gates and delays, most of the comparisons were done with the array multiplier. Based on the original recursive multiplier, a different implementation of the recursive multiplier is presented which has a "Dadda-like" structure, but has greater regularity since it is based on the initial base multiplier structure. The complexity is lower for the new design than the original recursive multiplier.

Problem

Dadda or Wallace fast multipliers provide high speed in calculations but they are difficult to layout because of their irregularity which also leads to larger area. Array multipliers (which may implement the modified Booth algorithm) have good regularity but do not have the speed of the fast multipliers. The recursive fast multiplier provides an interesting alternative because its performance is similar to a Dadda or a Wallace multiplier but it still maintains regularity. The problem associated with this method is its high complexity. This paper investigates the impact of the design of the base multiplier in the overall design and explores methods to reduce the complexity.

Introduction

In many engineering problems, the "divide-and-conquer" method is used to reduce large problems to smaller problems. This method can be applied in designing multipliers for large operand sizes based on breaking the multiplication into smaller pieces and using smaller multipliers. Many of the existing algorithms approach this method in two ways. One method is to break up the multiplication into a number of pieces, such as different rows of the multiplication matrix which are performed simultaneously [1]. Another method is to use a smaller multiplier at the initial stages and then combine the results. The recursive multiplier [2] falls into this category. Some multipliers that use this method have adders to sum up the results from the small multipliers[3]. Since these adders add delay of $O(n)$, this does not help the overall speed of the design. The designs that fall into the first group do not have the regularity and the simplicity of the multipliers as the other group. The Universal Multiplication Networks (UMN) presented in [4] also describes an iterative method which uses smaller multipliers to perform a large operand multiplication. Since some of the smaller multiplications are dependent on the carry outputs from other initial stages, the overall speed of the design is not optimized.

The recursive multiplier proposed in [2] is attractive because each doubling of the operand size increases the delay by a constant amount. The recursive multiplier is divided into two parts: the base multiplier and the reduction stage(s). The recursive multiplier operates by performing small base multiplications in parallel and then adding the results using reduction stages[2]. Depending on the size of the base multiplier, multiple levels of reduction may be needed. For example, to perform 16x16 multiplication with a 4-bit base multiplier, the first level of reduction generates results for 8-bit operands and the second level of reduction generates the result of the 16x16 multiplication. The overall delay of the recursive multiplier is $O(\log n)$, which is comparable to the delay of Dadda[5] or Wallace[6] multipliers, but the overall gate complexity is $O(n^2 \log n)$. With the bit-slice reduction network, the multiplier becomes very regular which allows the layout and the routing of the design to be much easier. This paper

0-7803-6514-3/00/$10.00©2000 IEEE

examines the impact of the base case multipliers and ways of reducing the overall gate complexity.

Approach

The recursive multiplier presented in [2] was examined to see what effect the type and size of the base multiplier has on the overall performance of the multiplier. Two different multipliers, the array multiplier and the Dadda multiplier, were studied as the base multiplier. The delay of the recursive multiplier was investigated as the size of the base multiplier was increased. The gate complexity was also compared.

The overall delay of the recursive multiplier can be separated into three different components: the delay of the base multiplier, the delay of the reduction stage(s), and the delay of the final addition. In analyzing the gate delay, the delay for the final addition was ignored since it is identical for all of the multipliers. For the base multiplier, the number of unit gate delays of a b-bit array multiplier, $\Delta_{A(b)}$, and a b-bit Dadda multiplier, $\Delta_{D(b)}$, are given by the following two equations :

$$\Delta_{A(b)} = 3b-2$$
$$\Delta_{D(b)} = 6\log_2 b - 5$$

where that the delay of either full or half adders is assumed to be 3 gate delays.

The delays for an n-bit recursive multiplier with a b-bit array base multiplier, $\Delta_{RM(n)\text{-}A(b)}$, and with a b-bit Dadda base multiplier, $\Delta_{RM(n)\text{-}D(b)}$, are shown below. The term $\log_2(n/b) = (\log_2 n - \log_2 b)$ gives the number of recursions required for given values of n and b

$$\Delta_{RM(n)-A(b)} = 1 + 3(b-1) + 9(\log_2 n - \log_2 b)$$
$$= 9\log_2(n/b) + 3(b-1) + 1$$

$$\Delta_{RM(n)-D(b)} = 1 + 6(\log_2 b - 1) + 9(\log_2 n - \log_2 b)$$
$$= 9\log_2 n - 3\log_2 b - 5$$

In comparing the gate complexity, the gate count for a b-bit array multiplier base, $G_{A(b)}$, is given by the following equation:

$$G_{A(b)} = b^2 + 12(b-2)(b-1) + 4(b-1)$$

where a full adder is assumed to require 12 gates and a half adder is assumed to require 4 gates.

For the recursive multiplier implemented with a b-bit array multiplier base, the gate complexity is not straightforward[2]. It was difficult to come up with a generalized formula for the gate count, but from the following equation, it can be seen that the gate complexity is $O(n^2 \log n)$.

$$G_{RM(n)-A(b)} = \left(\frac{n}{b}\right)^2 \bullet G_{A(b)} + (\log_2 n - \log_2 b)\left(\frac{n}{b}\right)^2 \sum C \bullet 2^{-x}$$

where x = 1 to $\log_2 n$ - $\log_2 b$

C is the constant for the number gates in the reduction stages.

The performance of the recursive multiplier is close to that of the Dadda multiplier but it comes at the expense of additional logic. The reduction cells adds logic to the recursive multiplier that is proportional to $O(n^2 \log n)$. However, in the reduction cell outlined in [2], each cell is maximally utilized only when all of the six inputs to the reduction cell are valid. When the base case is an array multiplier, none of the reduction cells in the first level of recursion make full use of the reduction cells (i.e., in the reduction cell shown in Figure 1, some of the primary inputs, C2, C1, C0, S2, S1, S0, are not connected to any outputs of the base multiplier). As a result, the first stage and the second stage of the reduction cells, which consist of 3 stages of reduction, can be modified by use of simplified reduction cells.

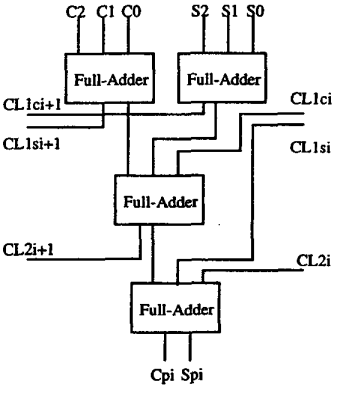

FIGURE 1: Reduction Cell Network [2]]

The modified reduction cells are shown in Figure 2. The selection of reduction cell depends on how many of the inputs are valid. For some of the reduction cells where only two inputs are valid (such is the case for the MSB bits), the first stage of full adders can be completely removed and the size of the reduction cells is reduced by half. For other scenarios, the initial two full adders of the first stage of the reduction can be modified by using a com-

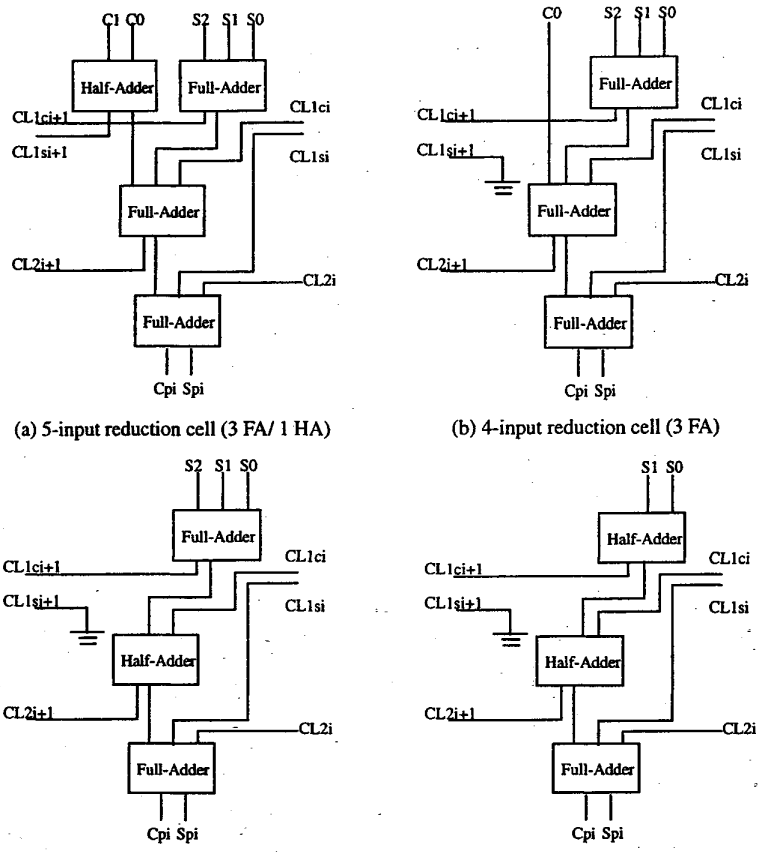

FIGURE 2: Different Reduction Cell Block Diagrams

bination of a single full adder and a single half adder. With careful layout of the new reduction cells, the final design should still remain very regular.

Another approach that can be taken to reduce the gate count further is to re-order the reduction. Instead of adding all four partial products of a multiplication, adding two partial products at a time results in the same delay, but reduces the gate count. In this new scheme, there are 2 levels of reduction as shown in Figure 3. The first level of reduction is a row of full adders and half adders. The second level reduction does a maximum of two stages of reduction of the carry-save inputs. The second level reduction is similar to the reduction cells illustrated in Figure 2 except that the first stage of adders is removed. This saves gates because it maximally utilizes the input to the second and the third stages of the original reduction cells. However, this type of reduction only benefits the first level of recursion. If multiple levels of recursion are needed, it becomes more difficult to implement this type of reduction after the first level.

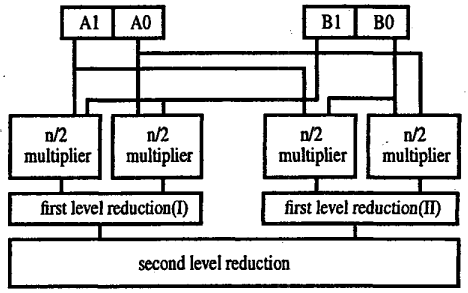

FIGURE 3: Modified recursive fast multiplier

Results

Using the equations for the delay of the different multipliers from the previous section, the gate delay was plotted against *n*, the operand size of the multiplier. In Figure 4, the different multipliers are compared. Comparing the recursive multiplier to the Dadda multiplier, there is not a big difference but the huge benefit can be seen when the

recursive multiplier is compared to the array multiplier. The recursive multiplier with the Dadda base multiplier is slightly faster than the one with the array base multiplier because of the faster base multiplier but the performance increase comes at the cost of extra area.

For the recursive multiplier with the array base multiplier, as the size of the base multiplier changes, the overall delay is greatly affected as shown in Figure 5. Since the array multiplier has an O(n) delay, as the base size increases, the base's delay becomes a bigger factor of the overall delay. However, with a recursive multiplier with a Dadda base multiplier, there is relatively little change in the delay as the base size increases, only a slight decrease in the delay as the size of the base size increases as shown in Figure 6. This is expected since both the Dadda and the recursive multiplier have O(log n) delay. From these plots, it is clear that the size of a Dadda base case multiplier does not have a great impact on the overall performance but if an array multiplier is used as the base multiplier, the selection of the size is critical to the overall performance.

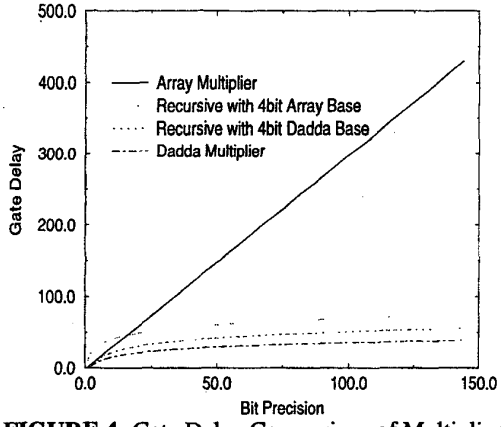

FIGURE 4: Gate Delay Comparison of Multipliers

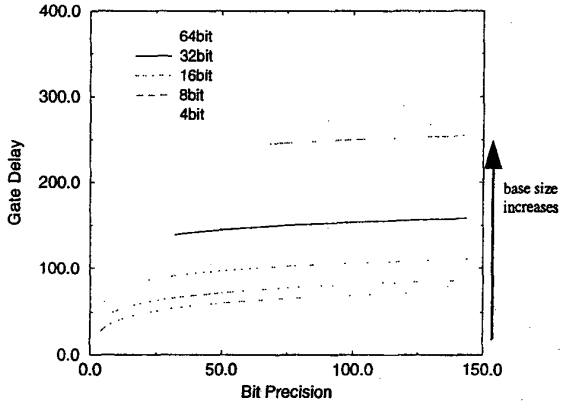

FIGURE 5: Recursive Multiplier Implemented with an Array Base Multiplie

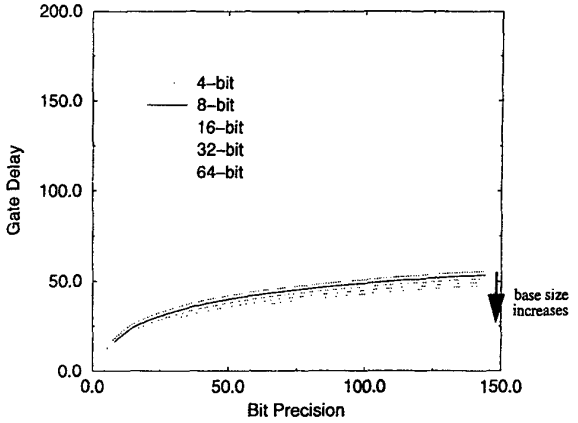

FIGURE 6: Recursive Multiplier Implemented with a Dadda Base Multiplier

By using the new approach of using different reduction cells, gate counts are reduced by approximately 25%. Taking advantage of "unused" inputs in the reduction cells (i.e., inputs that would have been grounded), simplifies the reduction cells and reduces the area. This saving in gates comes at the expense of having to design more cells, but the new cells are simple modifications of the original reduction cells, so they should not be difficult to design. However, the layout of the cells needs careful planning to assure that the different type of reduction cells can be easily be tiled. Figure 7 illustrates how each of the reduction cells should be layed out. Not all of the inputs/outputs will necessarily be valid but the same generic structure needs to be maintained.

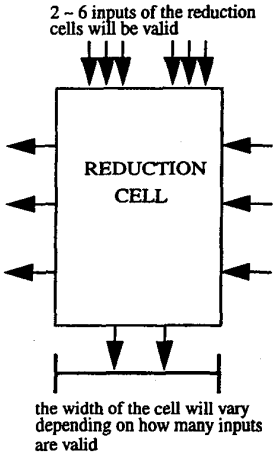

FIGURE 7: Layout for the Different Reduction Cell

The modified implementation of the recursive multiplier shown in Figure 3 results in an additional gate count reduction. The complexity of the different implementations

using an array multiplier are shown in Table 1 where the "modified recursive (I)" column represents data for recursive multiplier with modified reduction cells and "new modified recursive (II)" column presents data corresponding to Figure 3. The new modified recursive multiplier does present some additional overhead because not only is there initial routing from the base case multipliers to the first level reduction stage, there also needs to be additional routing between the first and the second levels of reduction. A similiar comparison is made for the different implementations using a Dadda multiplier in Table 2. The new modified recursive multiplier (II) is not implement using a Dadda base multiplier since the performance is not enhanced using this method. The first level of reduction needs to be done in a single reduction, but with Dadda base multipliers, two stages of reduction are needed to generate a number in a carry-save format. Hence, the new configuration illustrated in Figure 3 does not benefit a recursive multiplier with Dadda base multiplier.

Table 1: Comparison of Different 8-bit Multiplier Implementations (4-bit Array Base Multiplier)

	array multiplier	recursive multiplier	modified recursive (I)	new modified recursive (II)
half-adders	7	12	21	20
full-adders	42	64	45	39
gate delay	22	19	19	19

Table 2: Comparison of Different 8-bit Multiplier Implementations (4-bit Dadda Base Multiplier)

	Dadda multiplier	recursive multiplier	modified recursive (I)
half-adders	7	12	19
full-adders	35	52	41
gate delay	13	16	16

Conclusion

In multiplier design, there is often a trade-off between the complexity and the speed. Each design will have its own specific requirements which need to be satisfied by the design selection. Some of the trade-offs can be adjusted with the selection of different base multipliers. For example, in embedded applications where the absolute speed is not always the top priority, an array base multiplier might make more sense. To gain slightly more performance, a Dadda multiplier can be used as the base multiplier, but this increases the complexity. The modified reduction cells introduced in this paper reduce the complexity of the reduction stages.

For designs that only allow for short design times and if an existing design exists, the recursive multiplier seems to be a favorable choice. For example, if a 16 bit design exists for a multiplier and a 32 bit multiplier needs to be created, a 32 bit recursive multiplier can be designed with the existing 16 bit multiplier as the base case. Since design reusability is highly desirable, the recursive multiplier will be attractive.

The gate count measurement is probably not the best way of comparing silicon area. The array multiplier has a higher gate count compared to the Dadda multiplier, but the silicon area that an array multiplier occupies is smaller. Actual layout of the reduction cells needs to be done and incorporated into a recursive multiplier to make comparisons to other multipliers. Also, the reduced gate count can be expanded to multi-level recursive multiplication which will not be as simple as a single recursion multiplication.

Reference

[1] G.J., Hekstra and R. Nouta, "A Fast Parallel Multiplier Architecture," *1992 IEEE International Symposium on Circuits and Systems*, vol. 5, 1992, pp. 2128-2131.

[2] A. Danysh and Earl E. Swartzlander, Jr., "A Recursive Fast Multiplier," *Record of 32nd Asilomar Conference on Signals, Systems, and Computers*, 1998 pp. 197-201

[3] Behrooz Parhami, *Computer Arithmetic : Algorithms and Hardware Designs* New York: Oxford Univ. Press, 2000.

[4] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design,* New York: John Wiley, 1979.

[5] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, May 1965 pp. 346-356.

[6] C.S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Transactions on Electronic Computers,* vol. *EC-13*, Feb 1964 pp14-17.

[7] C.C. Stearns and Peng H. Ang, "Yet Another Multiplier Architecture," *IEEE 1990 Custom Integrated Circuit Conference*, 1990, pp. 24.6.1-4.