On the Implementation of a Three-operand Multiplier

Robert McIlhenny
rmcilhen@cs.ucla.edu

Miloš D. Ercegovac
milos@cs.ucla.edu

Computer Science Department
University of California
Los Angeles, CA 90024

Abstract

A new approach for a three-operand multiplier is proposed, using initial two-level Radix-4 recoding, in order to reduce the cost and delay of other utilized methods. A three-operand 4-bit multiplier is demonstrated as a model, and serves as a building block for three-operand multipliers of higher precision. The proposed method is shown to yield a significant reduction in both the cost and delay of a three-operand 4-bit multiplier.

1. Introduction

Much research has been done in the design of multi-operand addition [3] [7] and in parallel multiplication [1] [5] [8] [9]. In [2], the concepts of multi-operand addition and parallel multiplication are combined, and a new scheme is presented for the design of fast multi-operand multiplication.

A general m-bit multi-operand multiplier can be denoted in terms of the precision of the operands and the product as $(p_1, p_2, \ldots, p_m; q)$, where p_i is the operand length of input i, and q is the sum of the operand lengths:

$$q = \sum_{i=1}^{m} p_i \qquad (1)$$

In this paper, the emphasis will be on three-operand multiplication, in other words a $(p_1, p_2, p_3; q)$ multiplier. Three-operand multiplication occurs often in functions of the form: $f = ax * y$, where x and y are vectors, and a is a scalar. Other applications include calculating the linear address of a variable-length three-dimensional array, and calculating the determinant of a 3×3 matrix.

Three methods are considered in the implementation of a three-operand multiplier: (1) cascade method; (2) ROM method; and (3) proposed method. Analysis was done assuming operands of 4-bit precision. All multipliers are array multipliers, where cost is measured in terms of equivalent gates, as determined in [4], and delay is measured in terms of equivalent XOR gates.

2. Cascade Method

The cascade method consists of two multipliers in series. The first one multiplies two of the 4-bit operands to produce an 8-bit intermediate product. The 8-bit intermediate product is then multiplied by the third 4-bit operand to produce a 12-bit product. This is shown in Figure 1.

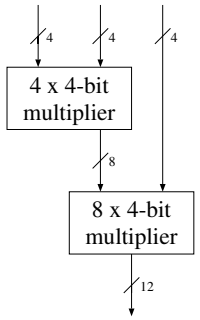

Figure 1. Cascade implementation of three-operand multiplier

Each multiplier can be divided into three stages: (1) partial product generation; (2) array reduction; and (3) final-level summation. Partial product generation can be perfomed by ANDing each of the bits of one operand with the other. Another method is Radix-4 Booth recoding, which reduces the number of partial product bits by half. Both methods are considered and evaluated for the cascade method. Array reduction is done using levels of (4:2) compressors, denoted as the Modified Wallace scheme in [6], which reduces the array to two words, and has lower reduction delay than the more conventional Wallace scheme consisting of

(3,2) counters. Final-level summation is performed using a 1-level 4-bit grouped Carry Lookahead Adder (CLA).

In generating the partial products, the non-recoding (ANDing) method for a general $m \times n$-bit multiplier, generates mn partial product bits. For the 4×4-bit multipler, this results in 16 partial product bits. Likewise, for the 8×4-bit multiplier, this results in 32 partial product bits. Thus, a total of 48 bits are generated, using 2-input AND gates. Assuming a 2-input AND gate has an estimated equivalent gate count of 2 and a delay of $0.5\delta_{XOR}$, the cost can be estimated as $2(48) = 96$ equivalent gates, and the delay as $1\delta_{XOR}$.

The Radix-4 Booth recoding method for a general $m \times n$-bit multiplier, assuming $m \geq n$, generates $(m+1)\lceil \frac{n}{2} \rceil$ partial product bits. For the 4×4-bit multiplier, this results in 10 partial product bits. Likewise, for the 8×4-bit multiplier, this results in 18 partial product bits. Thus, a total of 28 bits are generated, using recoder modules and multiplexers. Assuming the recoder has an equivalent gate count of 14 and delay of $1\delta_{XOR}$, and the multiplexer has an equivalent gate count of 6 and delay of $2\delta_{XOR}$, as detailed in [5], the cost can be estimated as $14(2) + 6(28) = 224$ equivalent gates, and the delay as $5\delta_{XOR}$.

In reducing the partial product array, the height of the original array is 4 for both the 4×4-bit and the 8×4-bit multiplier, when non-recoding is used. Assuming a (4:2) compressor, (3,2) counter, and a (2,2) counter have equivalent gate counts of 14, 7, and 5, respectively, and the delay of a (4:2) compressor is $3\delta_{XOR}$, this results in an equivalent gate count of $9(14) + 3(7) + 2(5) = 157$, and a delay of $2(3\delta_{XOR}) = 6\delta_{XOR}$. When recoding is used, there is no reduction stage, since the height of the array within each multiplier is 2.

For final level summation, assuming the non-recoding method is used, for the 4×4-bit multipler, two 4-bit words are to be added, and for the 8×4-bit multiplier, two 8-bit words are to be added. Assuming for a general n-bit CLA, the cost is $54(\frac{n}{4})$ equivalent gates and the delay is $(\frac{n}{4} + 2)\delta_{XOR}$, this results in an equivalent gate count of $54(3) = 162$, and delay of $7\delta_{XOR}$. The total cost of the cascade method using non-recoding is 415 equivalent gates, and the total delay is $14\delta_{XOR}$.

Assuming the recoding method is used, for the 4×4-bit multiplier, two 6-bit words are to be added, and for the 8×4-bit multiplier, two 10-bit words are to be added. Considering the same assumptions as before, and in addition assuming adding two 2-bit words has an equivalent gate count of 12, this results in a equivalent gate count of $54(3) + 12(2) = 186$, and a delay of $9\delta_{XOR}$. The total cost of the cascade method using recoding is 410 equivalent gates, and the total delay is $14\delta_{XOR}$.

3. ROM Method

The ROM method is presented in [2], consisting of utilizing the operands to address 256×8-bit ROM modules and producing the appropriate table-lookup result. At the first level, partial product words are referenced, and at the successive levels, array reduction is performed by referencing the appropriate partial sum at the address corresponding to decoding various partial product bits, until the actual product is obtained. This approach is shown in Figure 2.

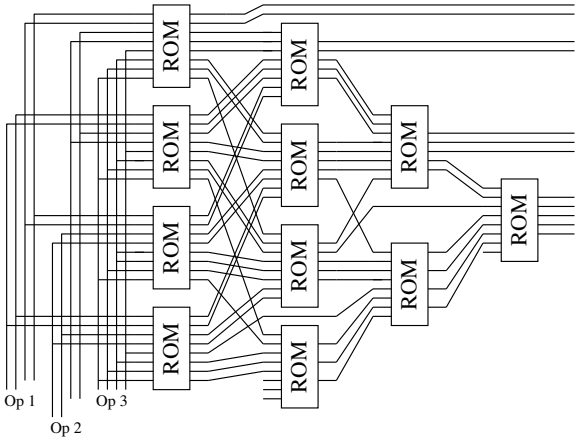

Figure 2. ROM implementation of three-operand multiplier

This approach consists of a 4-level network of eleven 256×8-bit ROM modules. Each ROM module decodes an 8-bit address into 1 of 256 values, and the referenced values are stored within registers. The cost of an individual 256×8-bit ROM module can be estimated as $6(256) + 1(2048) = 3584$ equivalent gates, and the delay can be estimated as $3\delta_{XOR}$. This results in a total equivalent gate count of $11(3584) = 39424$, and total delay of $4(3) = 12\delta_{XOR}$.

4. Proposed Method

The proposed method consists of initially producing all the partial product bits by means of two-level Radix-4 recoding. A simpler method is to use 3-input AND gates to AND together all the bits of the three operands. However, this would require $4 \times 4 \times 4 = 64$ 3-input AND gates to generate 64 partial product bits. This also creates an array of height 12, causing the array stage to have a relatively high cost as well. The details are not given, but the implementation cost of using 3-input AND gates to generate the partial product bits, and performing array reduction and final-level summation as before, is 636 equivalent gates, with a delay

of $14\delta_{XOR}$. This has neither an advantage over the other methods in terms of cost nor delay.

Initial two-level recoding is advantageous for three-operand multiplication in that it generates even fewer partial products than one-level recoding. One-level recoding reduces the number of partial products to about one-half, while two-level recoding reduces the number of partial products to about one-fourth. At the first stage of the proposed approach, the four bits of one operand are recoded, and the four bits of another operand are used to select the appropriate partial product bits. This generates two 5-bit words. At the second stage, the four bits of the third operand are recoded, and the bits of the two 5-bit words are used to select the appropriate new partial product bits. This generates four 6-bit words. Thus the total number of partial product bits generated is $4(6) = 24$. Assuming the same cost and delay of a recoder as stated for the cascade method, the cost of recoding for the proposed method is: $14(2) + 6(10) + 14(2) + 6(24) = 260$ equivalent gates, and the delay is $5\delta_{XOR}$.

The third stage consists of array reduction. Since the array is of height 4, this requires one level of (4:2) compressors, with a cost of 77 equivalent gates, and a delay of $3\delta_{XOR}$. The fourth stage consists of a 1-level 4-bit grouped CLA that operates over 6-bit input words. This has a cost of 66 equivalent gates, and a delay of $4\delta_{XOR}$. The total cost for the proposed method is 403 equivalent gates, and the delay is $12\delta_{XOR}$. The stages of the proposed approach are shown in Figure 3.

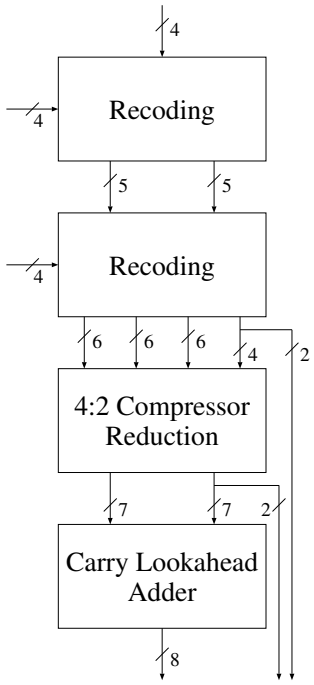

Figure 3. Proposed implementation of three-operand multiplier

5. Evaluation

The three methods for three-operand 4-bit multiplication are compared in terms of cost and delay. The results, considering both non-recoding (Non-rec) and recoding (Rec) generation of partial product bits, are shown in Table 5.

Method	PP Gen	Cost	Delay
Cascade	Non-rec	415	14
	Rec	410	14
ROM	——	39424	12
Proposed	Rec	403	12

Table 1. Comparison of 4-bit approaches

As is shown, the proposed method has the lowest cost and along with the ROM method has the lowest delay, making it an optimal approach for three-operand 4-bit multiplication.

6. Extension toward other multipliers

The methods presented for implementing three-operand 4-bit multiplication lead to the design of three-operand general n-bit multipliers, as well as general m-operand multipliers, which will be discussed below.

6.1. Three-operand n-bit multiplication

The methods discussed for three-operand 4-bit multiplication can be applied to general three-operand n-bit multiplication. The cascade method is implemented as a $n \times n$-bit multiplier and a $2n \times n$-bit multiplier in series. For general n-bit operands, assuming non-recoding, $n^2 + 2n^2 = 3n^2$ partial product bits are generated. Assuming recoding, $\lceil \frac{n}{2} \rceil (n+1)$ bits are generated by the first recoder, and $\lceil \frac{n}{2} \rceil (2n+1)$ bits are generated by the second recoder, for a total of $\lceil \frac{3n^2+2n}{2} \rceil$ bits. The height of the array for the first multiplier is $\lceil \frac{n}{2} \rceil$, as well as for the second multiplier. The first CLA operates over operands of length $2n - \lceil \log_2 n \rceil$, and the second CLA operates over operands of length $3n - \lceil \log_2 n \rceil$.

The ROM method for general three-operand n-bit multiplication grows exponentially with n. The network of ROM modules also grows more complex, Thus it is impractical for relatively large values of n.

The proposed method can be extended toward general three-operand n-bit multiplication in the same way for 4-bit operands, except that the cost is a function of $O(n^3)$.

Specifically, for general n-bit operands, the number of partial product bits generated by ANDing the operand bits is n^3. For two-level recoding the number is: $\lceil \frac{n^3+2n^2}{4} \rceil$, which is still $O(n^3)$. As n increases, the difference in the number of partial product bits generated by the cascade method and the proposed method is significant. For instance, considering $n = 16$, the cascade method generates 768 bits without recoding, and 400 bits with recoding. The proposed method, though, generates 1152 even with two-level recoding.

As the cost for the proposed becomes much larger than the cascade method as n increases though, the delay for the proposed method becomes significantly smaller than the cascade method. The comparison of cost and delay for the cascade and proposed methods for various values of n are shown in Figures 4 and 5, where the cascade method is assuming recoding.

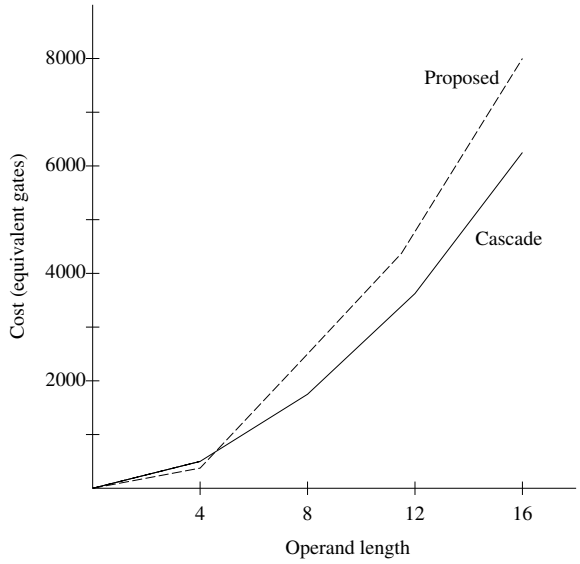

Figure 4. Comparison of Three-operand cost

6.2. General m-operand multiplication

The cascade method can be applied toward general m-operand multiplication, consisting of $(m-1)$ levels of multipliers, where at level k, $kn \times n$-bit multiplication is performed. This method generates $\frac{(n^2m(m-1)}{4} + \frac{n(m-1)}{2} = O(n^2m^2)$ partial product bits. The delay is a function of $O(m^2n)$, since each multiplier at level k utilizes a CLA that operates over approximately $(k+1)n$ bits, and $\sum_{k=1}^{m-1}(k+1)n = O(m^2n)$.

The proposed method can also be applied toward general m-operand multiplication, consisting of $(m-1)$ initial recoding levels, (4:2) compressor array reduction, and a final

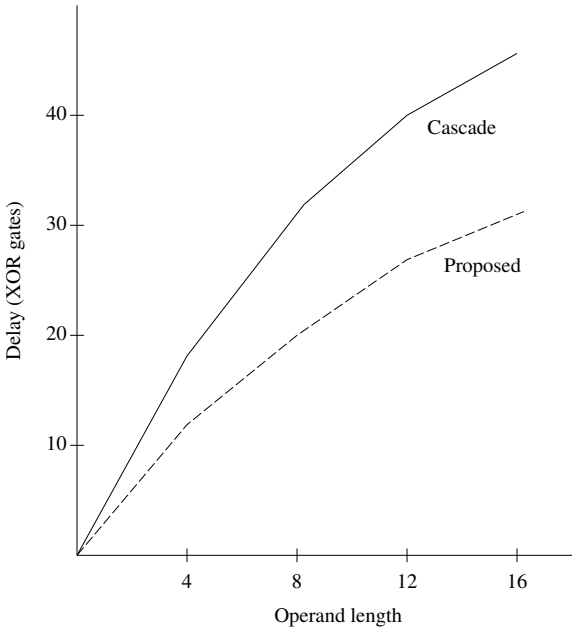

Figure 5. Comparison of Three-operand delay

level CLA. The number of partial products generated, however is $O(n^m)$, which for large m is not feasible. The delay though is of order $O(mn)$, so for feasible values of m, it can significantly reduce the delay over the cascade method.

7. Conclusion

A new approach for a three-operand multiplier has been presented. This approach utilizes two-level Radix-4 recoding to reduce the cost and delay of other utilized methods. This method can be extended as a building block for the implementation of general three-operand n-bit multipliers, as well as general m-operand n-bit multipliers. This approach is attractive for high speed computer arithmetic design.

Acknowledgments. This research has been supported in part by the NSF Grant MIP-9314172 "Arithmetic Algorithms and Structures for Low-Power Systems."

References

[1] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, May 1965.

[2] H. Kobayashi. A fast multi-operand multiplication scheme. *Proceedings of the 7th Symposium on Computer Arithmetic*, pages 246–250, March 1981.

[3] R. Lim. High-speed multiplication and multiple summand addition. *4th Symposium on Computer Arithmetic*, pages 149–153, October 1978.

[4] LSI Logic Corporation. *LCA500K Preliminary Design Manual*. 1994.

[5] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashimoto, H. Hayashida, and K. Maeguchi. A 10-ns 54x54-b parallel structured full array multiplier with 0.5-μm cmos technology. *IEEE Journal of Solid-State Circuits*, 26(4):600–605, April 1991.

[6] V. Oklobdzija and D. Villeger. Improving multiplier design by using improved column compression tree and optimizerd final adder in cmos technology. *IEEE Transactions on VLSI Systems*, 3(2):292–301, June 1995.

[7] S. Singh and R. Waxman. Multiple operand addition and multiplication. *IEEE Transactions on Computers*, C-22(2):113–120, February 1973.

[8] W. Stenzel, W. Kubitz, and G. Garcia. A compact high-speed parallel multiplication scheme. *IEEE Transactions on Computers*, C-26(10):948–957, October 1977.

[9] C. Wallace. A suggestion for a fast multiplier. *IEEE Transaction on Electronic Computers*, EC-13:14–17, February 1964.