High Radix Booth Multipliers Using Reduced Area Adder Trees

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

Abstract

The Reduced Area multiplier, the Wallace multiplier, and the Dadda multiplier each offer fast multiplication of signed binary numbers with the use of a large adder tree and a carry lookahead adder. However, their complexity makes them undesirable for some applications. A Booth Multiplier, on the other hand, offers simplicity and flexibility, by both breaking a multiplication up into pieces, and by allowing the size of the pieces to be chosen. Unfortunately, Booth multipliers become difficult to design for higher radices. The use of a fast adder tree, such as that found in a Reduced Area multiplier, permits straightforward design of very high radix Booth multipliers. Increasing the radix of a Booth multiplier in this manner results in large increases in speed with reasonable hardware cost.

1 Introduction

The Booth multiplication algorithm [4] provides a simple way to generate the product of two signed binary numbers $P = MR$. However, the algorithm is inherently serial and therefore not the fastest possible. Its attractiveness is that it consumes relatively little area when implemented. This is useful when speed is of lesser concern than hardware cost.

The choice of radix determines both the number of cycles required to generate the product and the complexity of the multiplier. A higher radix means fewer cycles, but greater complexity and time per cycle. On each cycle, some multiple of R is added to a register that is accumulating partial products. The value of M determines which multiple is used each cycle. As the radix increases, the algorithm selects from a larger and larger set of multiples of R, complicating the logic. These multiples may be generated before the algorithm is actually applied and the proper one simply selected each cycle [6].

However, if area is not important but speed is, a fast multiplier may be used. The Wallace and Dadda multipliers both generate all bit products simultaneously [5, 7]. A tree of full adders ([3,2] counters) and half adders ([2,2] counters) and a carry lookahead adder sums the bit products. The configuration of the adder tree differentiates the Wallace and Dadda approaches. The Reduced Area multiplier [2] makes use of an adder tree that is optimal in both area and speed. Nevertheless, all three multipliers consume much space when implemented.

Of course, there are many cases where speed is not essential and size is of greater concern. but a low radix Booth multiplier may be too slow. A Booth multiplier whose radix is above 16 or so becomes difficult to design due to the many multiples of a multiplicand R that may be required on any given cycle. A straightforward way of generating these multiples would simplify the design of efficient higher radix Booth multipliers.

The remainder of this paper explores a modified version of the basic Booth multiplier. The design is conventional except that instead of generating all multiples of R before beginning the algorithm, the multiples of R are generated as needed each cycle using an adder tree similar to that found in Reduced Area multipliers.

2 Approach

The overall architecture of the multiplier is shown in Figure 1. It is similar to a standard Booth multiplier. The R register holds operand R, while the P-M register holds the shifting operand M and accumulates partial products to form the final product P.

When the operands are loaded, the control logic decides, based on the low bit of operand M, whether the high part of the P-M register is loaded initially with zeroes or with operand R. Then, only on the first cycle, the low bit in the P-M register is assumed to be zero. Handling the low bit of M this way removes the need to pad M on the right with zeroes and add an extra cycle to the algorithm [6].

The number of bits from the low end of the M-P register examined each cycle is $log_2(radix) + 1$. These bits control which shifted versions of register R are fed into the adder tree each cycle. This is accomplished by using an array of AND gates (the addend matrix) in a way similar to standard array multipliers. The sum of the selected shifted versions of R is the absolute value of the desired multiple of R. The number of possible shifted versions is $log_2(radix)$, so that the adder tree must be able to handle $log_2(radix) + 1$ addends.

In the basic Booth algorithm, the highest of the control bits from the P-M register determines whether the multiple of R is added to or subtracted from the accumulating partial product. In this case, the bit actually controls whether the contents of the P-M register enter the adder tree complemented or uncomplemented. The result is that the output of the carry lookahead adder is either

1058-6393/95 $4.00 © 1995 IEEE

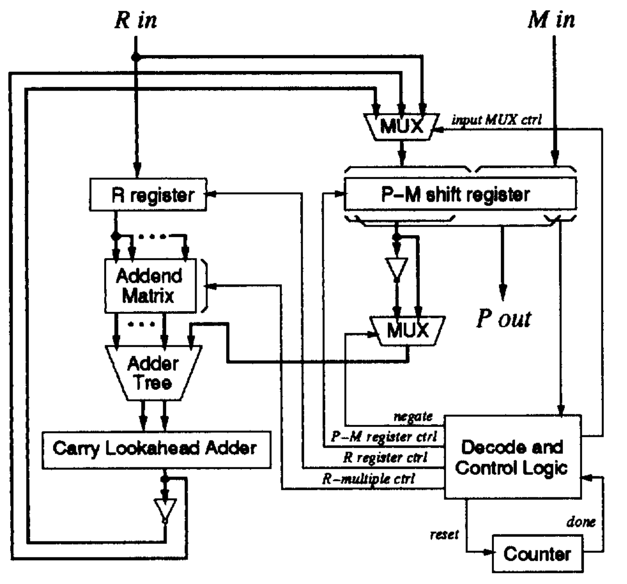

Figure 1: High level schematic of hybrid multiplier architecture

- the new high bits of the accumulating partial product, if the multiple of R would be added, or
- the complement of the new high bits of the partial product, if the multiple or R would be subtracted.

In the former case, the output of the adder is directly loaded into the P-M register, while in the latter case, the complement of the output is loaded. Hence, the P-M register always gets the correct value. Selectively complementing the output of the P-M register and the carry lookahead adder is simpler and faster in implementation than the algorithmic approach of negating R or its multiples.

Because the signs of the contents of both register R and register P-M can be either positive or negative, the addend matrix and the adder tree must handle signed values. This is accomplished in essentially the same way as in any fast multiplier, and does not complicate the analysis [1, 3]. The adder tree is assumed to be as in the Reduced Area multiplier. Optimizations that would be possible with detailed analysis are not attempted.

The longest delay path is from the P-M register through the decode and control logic, addend matrix, adder tree, carry lookahead adder, NOT gates, MUX, and back to the P-M register. This time is the minimum cycle time.

To investigate various configurations, a C program was coded. It calculates the relative delay and complexity of a set of multipliers, with word size and radix the only parameters varied. The designs include 16, 32, and 64 bit multipliers, and radices are powers of two varying from 2 to 65536. For each combination examined, the C program estimates the complexity of all parts of the multiplier as well as the delay of the

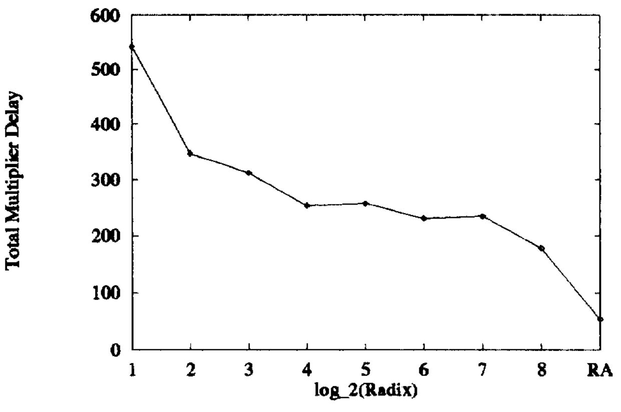

Figure 2: Delay of 16 bit multiplier for various radices.

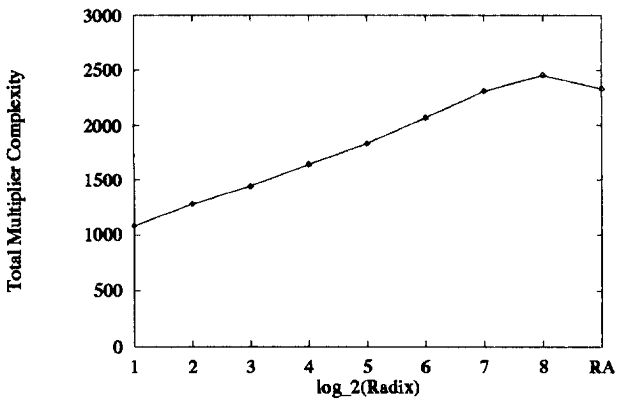

Figure 3: Complexity of 16 bit multiplier for various radices.

parts on the longest path. The Appendix tabulates assumptions about delay and complexity.

3 Results

The complexity and delay of each multiplier are compared in Figures 2 through 7. Each graph corresponds to one of the word sizes and shows the complexity or delay for all radix values examined for that word size. In addition, the rightmost point in each graph corresponds to a pure Reduced Area multiplier.

As expected, the higher the radix, the more complex the multiplier. Similarly, high radix multipliers tend to generate their results faster, both in terms of clock cycles and overall delay. However, even 3-cycle multipliers (where $log_2(radix)$ is half the word size) do not operate nearly as quickly as pure Reduced Area multipliers.

In order to more clearly explain the speed-area tradeoff in these multipliers, the product of the complexity and the delay of each multiplier is plotted (see Figures 8 through 10). The plots are normalized so

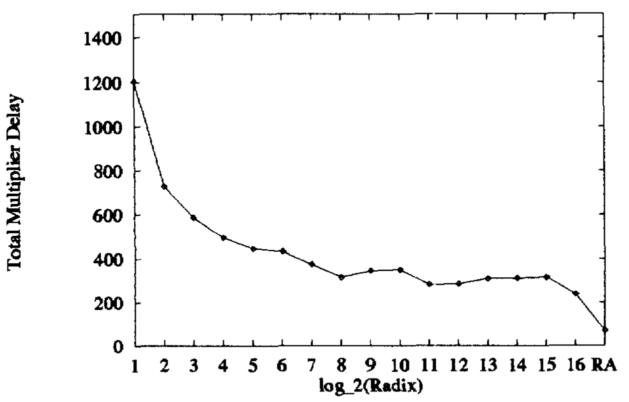

Figure 4: Delay of 32 bit multiplier for various radices.

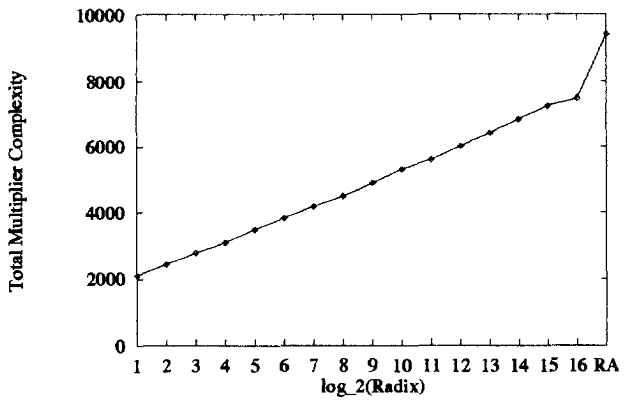

Figure 5: Complexity of 32 bit multiplier for various radices.

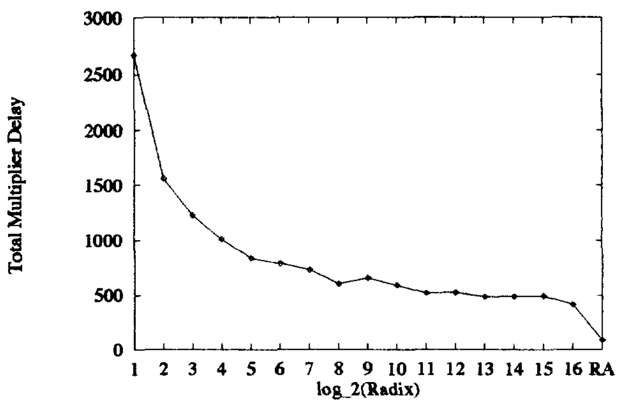

Figure 6: Delay of 64 bit multiplier for various radices.

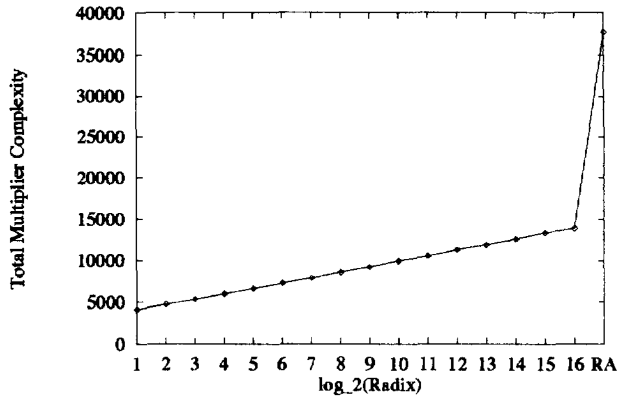

Figure 7: Complexity of 64 bit multiplier for various radices.

that the value corresponding to each radix two multiplier is 1000. For multipliers of a given word size, these numbers are a relative "unit of area per unit of speed" estimate and provide some indication of whether a given multiplier is an efficient use of space.

For the 16 bit case, higher radix multipliers do not provide much more speed per unit of area. However, for the 32 and 64 bit cases, multipliers with radices as high as 256 or so significantly improve the area/speed ratio. As radix is increased even more, however, the ratio tends to level off or even worsen. This is partly because increasing the radix does not always reduce the number of cycles required to do a multiplication with the Booth algorithm. Simultaneously, increasing the radix always increases the complexity of the control and addition logic.

4 Conclusion

A Reduced Area adder tree seems to be a good means of generating complex multiples of multiplicand R in a high radix Booth multiplier. Multipliers with radices as high as 256 or so seem to be a good option for 32 and 64 bit operands. Of course, the choice of radix in a given application depends on other factors, such as area constraints and clock period limitations. Furthermore, optimizations that would be unique to a configuration were not attempted and might bias the results somewhat differently.

Some area and delay assumptions were made to simplify the calculations of the C program. Better delay and area estimates for the carry lookahead adders and adder trees might be found, since optimizations unique to each configuration were not attempted. Furthermore, the control logic was not actually designed for all cases due to its complexity and because its area is only a small fraction of the whole multiplier. Its area and delay in the more complex multipliers were extrapolated from the simple cases. It is possible that the the speed and area of the control logic could benefit from optimization.

Multipliers for even larger words might profit from the approach outlined in this report. It is unlikely a designer would want to invest the resources necessary to multiply 128 bit operands together in a single cycle, although a high radix Booth multiplier might be ideal. Investigations involving other word sizes would probably confirm the utility of this approach to designing high radix Booth multipliers, especially if optimizations to the control logic and adder tree were considered.

References

[1] C. R. Baugh and B. A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, volume C-22, pages 1045-1047, 1973.

[2] K. C. Bickerstaff, M. J. Schulte, and E. E. Swartzlander. Reduced area multipliers. *Proceedings of the International Conference on Application-Specific Array Processors*, pages 478-489, 1993.

[3] P. E. Blankenship. Comments on 'A two's complement parallel array multiplication algorithm'. *IEEE Transactions on Computers*, volume C-23, page 1327, 1974.

[4] A. D. Booth. A signed binary multiplication technique. *Quarterly Journal of Mechanics and Applied Mathematics*, volume 4, pages 236-240, 1951.

[5] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, volume 34, pages 349-356, 1965.

[6] O. L. MacSorley. High-speed arithmetic in binary computers. *Proceedings of the IRE*, volume 49, pages 67-91, 1961.

[7] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, volume EC-13, pages 14-17, 1964.

Appendix

Tables 1 and 2 present the constants and equations used in estimating the total delay and total area of the multipliers examined in this study.

The delay and area of the register bits are based on the assumption that those on the critical path will need to select from several inputs each cycle. The delay and area equations for the carry lookahead adder are not exact but rather close approximations.

The delay equation for the control logic is extrapolated from simple cases, and if not exact, somewhat pessimistic. However, the effect of the control logic delay on the overall multiplier delay decreases as the radix increases.

To derive the complexity equation for the control logic, it can be observed that as more and more bits from the P-M register are to be examined, the logic added per bit increases somewhat linearly. By actually designing the logic for a few simple cases and extrapolating, this equation is developed. Although probably the roughest equation used, its effect on the results is not significant, as the control logic is a small fraction of the area of any of the multipliers.

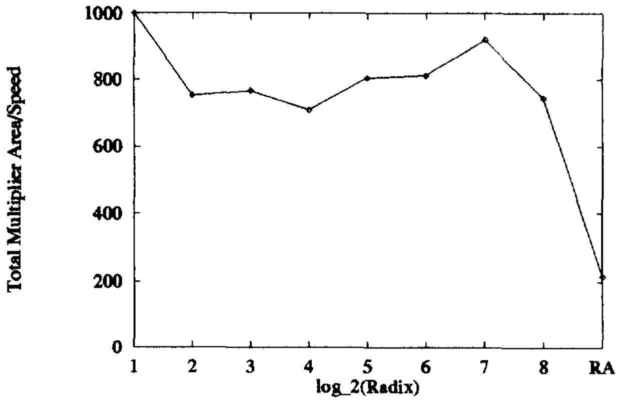

Figure 8: Normalized Complexity/Speed for 16 bit multiplier for various radices.

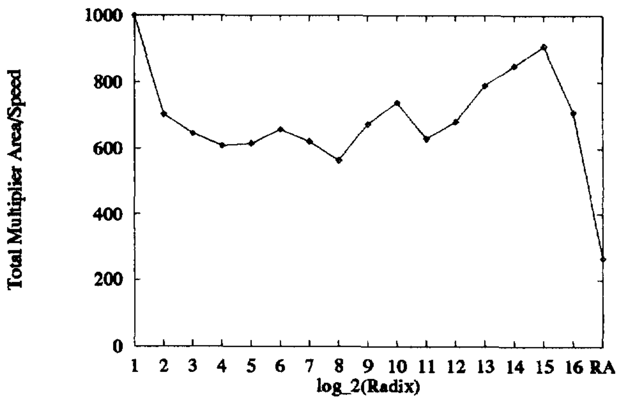

Figure 9: Normalized Complexity/Speed for 32 bit multiplier for various radices.

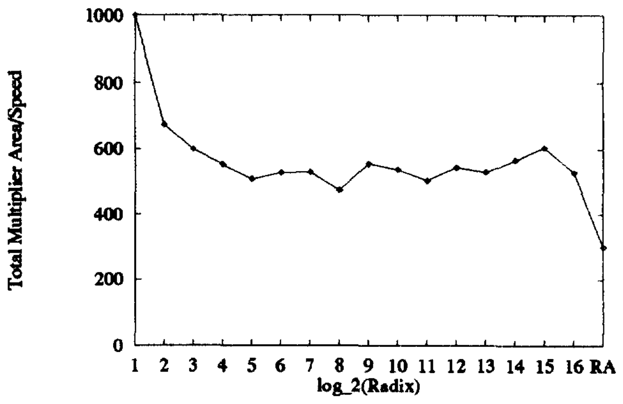

Figure 10: Normalized Complexity/Speed for 64 bit multiplier for various radices.

Logic elements	*Size*
Inverter	.6
2-input gate	1
3-input gate	1.8
4-input gate	2.6
Full adder	8.2
Half adder	3.6
D Flip-Flop	3.0
Register Bit	12.0
Area per Counter Bit	8.0
N-bit 2-way MUX	$0.6 + 3.0N$
N-bit 3-way MUX	$0.6 + 3.8N$
CLA Modified Full Adder	7.2
CLA4 block	25.2
N-bit CLA (Radix 4)	$7.2N + 25.2(N/3)$
Radix N Multiplier Control Logic	$20+ 2(log_2 N)(log_2 N + 1)$

Table 1: Areas assumed for the calculations

Logic elements	*Delay*
Inverter	.8
2-input gate	1
3-input gate	1.6
4-input gate	2.3
Full adder	5.6
Half adder	2.8
D Flip-Flop	3
Register Bit	7.6
N-bit 3-way MUX	2.6 (data thru)
N-bit CLA (Radix 4)	$9.2 log_4 N - 1.5$
Radix N Multiplier Control Logic	$2.5 + log_2 N$

Table 2: Delays assumed for the calculations