

A Modified Booth Algorithm for High Radix Fixed-Point Multiplication

Rajendra Katti

Abstract—The Booth multiplication algorithm produces incorrect results for some word sizes, when it is extended for higher radix, fixed-point multiplication. We present a modification of the Booth algorithm that produces correct results when the radix is any power of 2 and the multipliers are of any size.

Index Terms—Algorithm, fixed-point multiplication, higher-radix multiplication, multiplication, signed numbers.

I. INTRODUCTION

Andrew Booth presented an algorithm to multiply two signed, two's complement numbers [1]. This algorithm has been extended for higher radix operands for computational speedup. However, for certain size operands, correction shifts have to be performed on the product produced by the Booth algorithm in order to obtain a correct product. This paper presents a modification of the Booth algorithm that does not need any correction shifts for any operand size and for any radix that is a power of 2. In [2] a modified Booth algorithm is presented for operand sizes that are a power of 2. This modified algorithm requires correction cycles after the original Booth algorithm has completed. The algorithm presented in this paper is not restricted to operand sizes that are powers of 2 and it requires no correction cycles. Section II describes the standard Booth multiplication algorithm and Section III describes the modified Booth algorithm. Section V explains the modified algorithm via two examples.

II. MULTIPLICATION USING RADIX-2 BOOTH RECODING

The rules for standard radix-2 Booth recoding are as follows ([1], [2]):

- 1) Append a zero to the right of the LSB of the multiplier number, A .
- 2) Inspect groups of two adjacent bits of A , starting with the LSB and the appended zero.
 - if the pair is 00 or 11, then shift the partial product one bit to the right.
 - if the pair is 01, then add the multiplicand B to the partial product and shift the new partial product one place to the right.
 - if the pair is 10, then subtract B from the partial product and shift the new partial product one place to the right.
- 3) Proceed with overlapping pairs of bits such that the MSB of a pair becomes the LSB of the next pair. In this manner one bit of the multiplier number, A is eliminated in each pass through the algorithm.
- 4) When the last pair of bits is examined, the partial product is updated following the rules in step 2 above except that no shift is performed.

Manuscript received October 26, 1993; revised April 18, 1994 and July 20, 1994.

The author is with the Department of Electrical Engineering, North Dakota State University, Fargo, ND 58105 USA.
IEEE Log Number 9405517.

III. THE MODIFIED MULTIPLICATION ALGORITHM

Let $|A|$ denote the number of bits in a binary number A . Then the modified Booth multiplication algorithm is:

- 1) Append a zero to the right of the LSB of the multiplier number, A . The appending of the zero does not imply the use of $|A| + 1$ bit registers. It implies that the first pair of bits inspected has a least significant bit of 0.
- 2) Inspect groups of two adjacent bits of A , starting with the LSB and the appended zero.
- 3) Form a partial product P such that,

$$|P| = |A| + |B| - 1$$

where A is the original multiplier and B the multiplicand. The initial value of P is zero.

- if the pair is 00 or 11, then shift the partial product one bit to the right.
- if the pair is 01, then sign extend B to form C such that,

$$|C| = |A| + |B| - 1 - (i - 1)$$

where i is the pass number through the algorithm. The first pass through the algorithm is numbered 1. C is then added to the partial product and the new partial product is shifted one place to the right.

- if the pair is 10, then sign extend $-B$ (two's complement of B) to form D such that,

$$|D| = |A| + |B| - 1 - (i - 1)$$

D is then added to the partial product and the new partial product is shifted one place to the right.

- 4) Proceed with overlapping pairs of bits such that the MSB of a pair becomes the LSB of the next pair. In this manner one bit of the multiplier number, A is eliminated in each pass through the algorithm.
- 5) When the last pair of bits is examined, the partial product is updated following the rules in part 2 of step 3 above except that no shift is performed.

In the above algorithm the partial product actually refers to the existing sum of partial products and all shifts are performed after sign extension. The above algorithm can be extended to any radix r that is a power of 2 by making the following changes. In part 2 of step 3 above compute C and D such that

$$|C| = |A| + |B| - 1 - (i - 1) \log_2 r$$

$$|D| = |A| + |B| - 1 - (i - 1) \log_2 r$$

For a higher radix, C and D are not just added to the partial product as in part 2 of step 3 above. For example if the radix is 4, then 3 bits of the multiplier are considered at a time. If the bits are 001 or 010 then the new partial product is formed by adding C to the current partial product and shifting the partial product right twice. The radix 4 and radix 8 recoding rules have been described in [2]. If the radix is 4 two bits of the multiplier are eliminated every cycle resulting in a faster multiplier.

IV. CORRECTNESS OF THE ALGORITHM

If the radix is 4, three bits are inspected in each cycle and two bits are eliminated from the multiplier in each pass of the algorithm. Let us assume that the multiplier has an odd number of bits n after the binary point. Then the number of passes through the algorithm is $(n+1)/2$. Let the number of bits to the right of the binary point in the multiplicand be m . Let us assume that m is even. In this case the original Booth algorithm will always result in a product that has an even number of bits to the right of the binary point. This is because the number of passes in the algorithm is $(n+1)/2$ which is even and the number of bit-shifts in each pass is 2. The result of the original Booth algorithm is incorrect as the correct product has an odd number of bits to the right of the binary point. The correct product actually has $(n+m)$ bits to the right of the binary point. This problem can be solved if the partial product in the i^{th} pass is sign extended to contain $(n+m-(i-1)\times 2)$ bits. This ensures that the product size is correct and therefore results in the correct answer. If the radix is r , the number of bits eliminated in each pass is $\log_2 r$. Therefore it is sufficient to sign extend the partial product in the i^{th} pass such that it contains $(n+m-(i-1)\times \log_2 r)$ bits. In the lemma below we assume that the multiplier A has n -bits to the right of the binary point and the multiplicand B has m -bits to the right of the binary point. We also assume that both A and B have only one bit to the left of the binary point.

Lemma If the following equation is true then the original Booth algorithm is correct.

$$n = (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

Proof: Since the radix is r , the multiplier A is shifted right $\log_2 r$ bits through every pass of the algorithm, except the last pass. The number of passes in the algorithm is,

$$\lceil (n - \log_2 r + 1) / \log_2 r \rceil + 1,$$

as $\log_2 r + 1$ bits are inspected in each pass of the algorithm. Since the last pass involves no shift, the number of bits to the right of the binary point in the product must be,

$$m + (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

But we know that the number of bits to the right of the binary point must be $(n+m)$. Therefore the original Booth algorithm will work when the following equation is true,

$$n = (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

Our modification makes sure that the number of bits in the partial product of each pass is $(n+m)$. This results in the correct final product.

Another way to produce the right product is to change n such that the following equation is satisfied.

$$n = (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

This will result in an increase in n . It does not however mean that the size of the registers used will increase. All it means is that the number of passes through the algorithm will increase. We need to add another step to the original Booth algorithm before step 1. Let us call this step 0 (shown below).

Step 0. Append zeros to the right of the multiplier such that the total number of bits to the right of the binary point 'n' satisfies the following equation.

$$n = (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

In the above equation r is the radix.

As an example consider the multiplier 0.101 with radix 4. Since the number of bits to the right of the binary point is 3, we have $n = 3$. Since the following equation is not satisfied,

$$n = (\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r$$

we change the multiplier to 0.1010. This makes $n = 4$, and the above equation is satisfied. This would result in 3 passes through the algorithm instead of 2.

The following general rule tells us how many zeros to append to the multiplier for the equation in the Lemma to be satisfied.

Rule: The number of zeros appended is such that the total number of bits to the right of the binary point is the smallest integer, greater than n that is divisible by $\log_2 r$. We have assumed that the radix is r and the number of bits to the right of the binary point in the original multiplier is n .

The above rule will change n for the multiplier such that the equation stated in the Lemma is satisfied. For example if the multiplier is 0.111101, and the radix is 8, then the new multiplier is 0.11110100. The number of zeros appended is such that the total number of bits to the right of the binary point is 9 (note that 9 is the smallest integer divisible by 3 that is larger than 7). This new multiplier with the original Booth algorithm will result in the right product.

Let us consider the case when the number of bits in the multiplier and the multiplicand is some power of 2, say 2^a . This implies that the number of bits to the right of the binary point is $2^a - 1$ as one bit is to the left of the binary point. When $n = 2^a - 1$, and $r = 4$,

$$(\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r = 2^a - 2 \neq n.$$

This implies that the original Booth algorithm will not work. However if we append one zero to the multiplier we obtain $n = 2^a$ and

$$(\lceil (n - \log_2 r + 1) / \log_2 r \rceil) \log_2 r = 2^a = n.$$

This implies that Booth's original algorithm will work.

V. EXAMPLES

We shall demonstrate our algorithm with 2 examples.

Example 1: Let $A = 0.101 = \frac{5}{8}$, and $B = 0.11 = \frac{3}{4}$. Let the radix equal 4. Using the original Booth algorithm for radix 4 we obtain the product of A and B to be 0.1111, which is incorrect [2]. Using our algorithm we first append a zero to A to obtain 0.1010. The initial partial product P_0 is 000000 as $|A| = 4$ and $|B| = 3$. Since the last 3 bits of the new A are 010 we form $C = 000011$. Notice that B has been sign extended to form C . Since this is the first pass through the algorithm, i is 1. C is then added to the partial product to form a new partial product $P_1 = 000011$. P_1 is then shifted right twice. Two bits are then eliminated from A and the next three bits of A to be considered are 010. The new C is now 0011. C has again been obtained by sign extending B . This is the second pass through the algorithm and i is 2. The new partial product is now $P_2 = 001111$. Notice that none of the partial products have a binary point. The binary point is simply inserted into the final partial product after the MSB. Therefore the final partial product is 0.01111. This is the correct product of A and B .

If we were to change the multiplier A to 0.1010, then after going through 3 passes of Booth's original algorithm for $r = 4$, we would obtain the product as 0.011110. Notice that the number of bits to the right of the binary point in the product is 6.

Example 2: Let $A = 1.11 = \frac{-1}{4}$, and $B = 0.111 = \frac{7}{8}$. Let the radix equal 8. Using the original Booth algorithm for radix 8 we obtain the product of A and B to be 1.001, which is incorrect [2]. Using our algorithm we first append a zero to A to obtain 1.110. The initial partial product P_0 is 000000 as $|A| = 3$ and $|B| = 4$. Since the last 4 bits of the new A are 1110 we form $D = 111001$. Notice that B has been sign extended to form D . Since this is the first pass through the algorithm, i is 1. D is then added to the partial product to form a new partial product $P_1 = 111001$. The final product is therefore 1.11001.

VI. CONCLUSION

We have presented a modification to the Booth multiplication algorithm. This modification produces correct results for higher radix, fixed point multiplication when the radix is a power of 2. Unlike previous modifications our modification eliminates all correction cycles and is applicable to any operand sizes.

REFERENCES

- [1] A. Booth, "A signed binary multiplication technique," *Q. J. Mech. Appl. Math.*, vol. 4, pp. 236-240, 1951.
- [2] P. E. Madrid, B. Millar, and E. E. Swartzlander, "Modified Booth algorithm for high radix fixed-point multiplication," *IEEE Trans. VLSI Syst.*, vol. 1, no. 2, pp. 164-167, June 1993.