A Review on Optimizing Efficiency of Fixed Point Multiplication using Modified Booth’s Algorithm

Mahendra R. Bhongade, Manas M. Ramteke, Vijay G. Roy

Author Details
Mahendra R. Bhongade, Department of Electronics, R. C. E. R. T., Chandrapur, R. T. M. Nagpur University, India, E-mail: mrbhongade@rediffmail.com
Manas M. Ramteke, Department of Electronics, R. C. E. R. T., Chandrapur, R. T. M. Nagpur University, India, E-mail: manasramteke87@gmail.com
Vijay G. Roy, Department of Electronics, R. C. E. R. T., Chandrapur, R. T. M. Nagpur University, India, E-mail: vgroy15@yahoo.com

Keywords
Modified Booth’s Algorithm, fixed point multiplication, high radix, correction cycle, encoder, signed number, partial product.

ABSTRACT

High speed fixed point multiplication becomes significant constraint for efficient operation of low power VLSI devices. Andrew Booth presented an algorithm to multiply two signed, two’s complement numbers. 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. Many researchers tried and tested innovative and efficient way to solve the restrictions of Booth’s algorithm. In this paper, we have presented modified approach towards reduction of correction cycles while evaluating multiplication. This paper also demonstrates the method and set of rules associated with this algorithm.
1. INTRODUCTION

As the need for faster computing increases the need for exploring ways to quickly and accurately perform multiplication also increases. Booth’s standard radix 2 algorithm for the multiplication of two two’s complement numbers cannot be extended to higher radix multiplication without modification. Higher radix recoding is a powerful method of speeding up the multiplication process and it is therefore useful to explore ways to extend the Booth multiplier to these higher radices.

Many multiplication algorithms exist which increase the speed of operation over the classic shift and add method. These algorithms may be divided into two categories: variable shift methods and uniform shift methods. The variable shift methods are disadvantageous for clocked systems since the time required for a multiply is data dependent. Booth’s algorithm, a uniform shift method, examines two bits of the multiplier at a time to determine the correct multiple of the multiplicand to be added to the partial product. This method requires no sign correction for a two's complement number and the decoding of the multiplier may be begun from either direction. The major disadvantage of the algorithm is that the process still requires ‘n’ shifts and an average of ‘n/2’ additions for an ‘n’ bit multiplier.

2. MULTIPLICATION USING RADIX-2 BOOTH RECODING

The rules for standard radix-2 Booth recoding are as follows:
   a) Append a zero to the right of the LSB of the multiplier number, A.
   b) Inspect groups of two adjacent bits of A, starting with the LSB and the appended zero.
      i. If the pair is 00 or 11, then shift the partial product one bit to the right.
      ii. 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.
      iii. If the pair is 10, then subtract B from the partial product and shift the new partial product one place to the right.
   c) 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.
   d) When the last pair of bits is examined, the partial product is updated following the rules in step (b) above except that no shift is performed.

3. THE MODIFIED MULTIPLICATION ALGORITHM

Let |A| denote the number of bits in a binary number A. Then the modified Booth multiplication algorithm is:
   a) 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.
   b) Inspect groups of two adjacent bits of A, starting with the LSB and the appended zero.
   c) 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.
      1. If the pair is 00 or 11, then shift the partial product one bit to the right.
      2. 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.
      3. If the pair is 10, then sign extend ‘-B’ (two’s complement of B) to form D such that, D is then added to the partial product and the new partial product is shifted one place to the right.
   d) 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.
   e) When the last pair of bits is examined, the partial product is updated following the rules in part 2 of step (c) 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 (c) 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 (c) 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.

We shall demonstrate our algorithm with examples.

Example 1: Let $A = 0.101 = 5/8$, and $B = 0.11 = 3/4$. 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.

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 4 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 $B$.

Fig.1 shows the basic structure of modified Booth algorithm where partial Product generation mechanism is prescribed. In this structure multiplication is carried out between multiplicand $X$ and multiplier $Y$. Encoder is used to convert multiplier bits into Booth’s encoding bits. Partial product generator takes encoding bits of multiplier and generates partial products in such that that the carry out generated in each product term should propagate in its sequential order so that it could be properly transferred towards carry propagate adder. Finally carry propagate adder Matrix combine all propagated carry outs in their sequential positions such that Final product adder would add them together with the partial products to generate final result of multiplication.
**TABLE 1**
Partial Product Generation using Modified Booth Algorithm

<table>
<thead>
<tr>
<th>Multiplier bits</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y_{n-1} Y_{n-3}</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2}</td>
</tr>
<tr>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2} + X</td>
</tr>
<tr>
<td>0 1 0</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2} + X</td>
</tr>
<tr>
<td>0 1 1</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2} + 2X</td>
</tr>
<tr>
<td>1 0 0</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2} - 2X</td>
</tr>
<tr>
<td>1 0 1</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2} - X</td>
</tr>
<tr>
<td>1 1 0</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2}</td>
</tr>
<tr>
<td>1 1 1</td>
<td>PP_{i} \leftarrow (1/4) PP_{i+2}</td>
</tr>
</tbody>
</table>

For i= (n-1), (n-3),......., 3, 1

PP_{i} = ith partial product, PP_{n+1}=0

X=multiplicand

Y=multiplier, n+1bit wide, Y_{n}=0

4. **APPLICATION OF MODIFICATION RULE TO IMPLEMENT HIGH RADIX MULTIPLIERS**

The form of the rule presented in is adequate to understand the basic problem inherited in the Booth algorithm in high radix multiplication. However, it does not shed light on how the modification is used in multipliers with operand sizes of 8, 16, 32, 64 bits etc. In other words, all the examples thus far presented show how the multiplication would be done on paper, not how it would be done in a hardware implementation which would always have fixed length words and the possibility of many trailing zeros in the register where words are stored. For example, if one was implementing a 32 bit multiplier utilizing radix 8 bit recoding, it should be known beforehand the correction cycle required without having to dynamically determine number of significant digits required and produced. As will be shown, the correction cycles needed follow a regular pattern for each recoding and multiplier size.

5. **CONCLUSION**

As the need for faster computing increases, the need for exploring ways to quickly and accurately perform multiplication also increases. A simplified proof of a modification of Booth’s multiplication algorithm to a form which examines three multiplier bits at a time is presented. In comparison with the original Booth’s algorithm, which examines two bits at a time, the modified algorithm requires half the number of iterations at the cost of somewhat increased complexity for each iteration. The modified Booth multiplier works with not only 32-bit but also two 16-bit or four 8-bit data at one clock, ensures the speed and saves the chip area at the same time.

**Acknowledgment**

The authors would like to acknowledge their colleagues for their continuous support and positive encouragement. They also thank their Principal, Dr. K. R. Dixit for providing immense facilities and excellent academic infrastructure.
References