Fast Redundant Binary Partial Product Generators for Booth Multiplication

Bijoy Jose and Damu Radhakrishnan
Department of Electrical and Computer Engineering
State University of New York
New Paltz, New York, USA 12561
bijoyaj@gmail.com, damu@engr.newpaltz.edu

Abstract—The use of signed-digit number systems in arithmetic circuits has the advantage of constant time addition irrespective of word length. In this paper, we present the design of a binary signed-digit partial product generator, which expresses each normal binary operand in one's complement form with an extra bit denoting the sign bit of the operand. The carry free nature of RBAs is exploited to add the extra bits with the partial products, without using additional adder stages. This technique allows RB encoding to be used in multipliers with operand widths which are perfect powers of two, without increasing the delay of partial product accumulation. The proposed partial product generator achieves the highest reduction in the number of partial products for a radix-4 multiplier (78%), by combining advantages of RB encoding with RB addition. The proposed partial product generator together with a set of fast redundant binary adder stages configured in a binary tree fashion can result in the design of high performance multipliers.

I. INTRODUCTION

The fast growth in computing power has enabled us to develop sophisticated algorithms and perform complicated functions. It also resulted in a requirement to increase processing power even more, in order to do these operations more efficiently. In this regard, researchers have always been trying to develop faster and faster computational hardware. This demands the development of faster arithmetic circuits, such as adders, multipliers etc.

Many adder and multiplier designs have been proposed in the past to satisfy various objectives such as speed, area and power. Even though earlier designs focused more on minimizing the Silicon area, the focus has shifted more towards speed and power in the last decade. The use of signed digit arithmetic for the design of adders is worth mentioning in this regard. Signed digit arithmetic has its property, which makes it possible to have constant time addition, thereby eliminating the perpetual problem of carry propagation during addition. Redundant binary (RB) number systems are used to perform signed digit arithmetic in binary [1]. In an RB number system with the digit set [-1, 0, 1], each digit is coded with two bits. An RB digit $X_i$ can therefore be represented by the bit pair $(X_i^-, X_i^+)$. The constant time addition property has spurred interest among researchers to perform addition and multiplication of normal binary (NB) operands in RB number systems. Special conversion circuits have also been designed to enable fast NB to RB and RB to NB conversion.

II. MULTIPLIERS

A multiplier essentially consists of two operands, a multiplicand ‘Y’ and a multiplier ‘X’, and produces a product ‘P’. In a conventional multiplier, a number of partial products are formed first by multiplying the multiplicand with each bit of the multiplier. These partial products are then added together to generate the product ‘P’. In short, we can break down multiplication into two parts, namely partial product generation and partial product accumulation. Speeding up multiplication therefore must aim (i) speeding up partial product generation (PPG), (ii) reduce the number of partial products, (iii) speeding up partial product summation or (iv) a combination of one or more of the above.

An algorithm to reduce the number of partial products was first proposed by Booth [2]. The Booth algorithm was based on the fact that only fewer partial products need to be generated for a multiplier consisting of consecutive ones or zeros. An efficient algorithm was later proposed by McSorley, known as the modified Booth algorithm or radix-4 Booth multiplication, which reduced the number of partial products by half [3].

The next step involves the addition of these partial products in the fastest manner possible. Speeding up the addition of partial products required faster adders. The major problem with fast addition was carry propagation. This spurred lot of interest in the design of arithmetic circuits using RB number system. One of the early works in this field is a high-speed multiplier using a tree of redundant binary adders by Harata et al. [4]. The addition of two RB numbers was performed in constant time in a two-step method. A Redundant Binary Adder (RBA) cell was defined and a
16x16-bit high-speed multiplier using 2’s complement binary numbers was implemented. The NB digits were converted into RB during partial product generation. A 50% reduction was achieved in the number of partial products. Further reduction in the number of partial products was obtained by Makino et al. [5] by using radix-4 Booth algorithm and by pairing the partial products to form single ones. Furthermore, a modified version of an efficient RB to NB converter proposed by Yen et al. [6] was used in their design. A new RBA cell was also defined by Makino et al. to attain high speed addition. Besli et al. used the above RBA cell to design a 54x54-bit multiplier based on a RBSD radix-8 Booth encoder [7,8]. The number of partial products was reduced to 66% in their design. A 54x54-bit radix-64 multiplier using the least number of transistors was designed by Lee et al., which expressed each partial product as a combination of y, 2y and 3y, where y is the multiplicand [9]. The computation of 3y created an overhead in the partial product generation block.

Among the available multiplier designs, Makino multiplier achieved the greatest reduction in the number of partial products using their Redundant Binary Partial Product Generator (RBPPG) [5]. In their design, the sum of two partial products A and B represented in two’s complement form is converted to RB notation by a simple grouping of the bits in A and B. This is illustrated as follows:

\[ A + B = A - B - 1 \]  

(1)

where \( \overline{B} \) is the one’s complement of \( B \). Equation 1 can be rewritten as:

\[ A + B = (A, \overline{B}) - 1 \]  

(2)

where \( (A, \overline{B}) = A - \overline{B} \).

Using the RB Encoding 1 given in Table I,

\[ A + B = (A, \overline{B}) + (0,1) \]  

(3)

One of the objectives of the above approach was to avoid the time consuming two’s complement addition. The sum was obtained in constant time, at the expense of a few inverters, with the added advantage of converting the result in RB form. But this does not cancel the delay in obtaining negative NB partial products from the Booth encoder in two’s complement form. The negative NB partial products are obtained by adding 1 to the one’s complement of the numbers. The resulting carry propagation introduces extra delay during partial product generation. A second delaying factor is due to the non-unique coding of ‘0’ as seen in Table I. Because of this, every (1,1) pair generated has to be reconverted back to a (0,0) pair in the Makino RBA. Kim et al. [10] offered a solution for the negative NB representation by the use of an alternate RB encoding and an error-correction word. This alternate RB Encoding 2 is shown in Table I, which defines \( (A, B) \) as \( (A, B) = A - \overline{B} \).

### Table I. Redundant Binary Encoding

<table>
<thead>
<tr>
<th>RB digit</th>
<th>Encoding 1</th>
<th>Encoding 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X_i )</td>
<td>( (X_i^*, X_i') )</td>
<td>( (X_i^*, X_i') )</td>
</tr>
<tr>
<td>0</td>
<td>(0,0)</td>
<td>(0,1)</td>
</tr>
<tr>
<td>1</td>
<td>(1,0)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>-1</td>
<td>(0,1)</td>
<td>(0,0)</td>
</tr>
<tr>
<td>0</td>
<td>(1,1)</td>
<td>(1,0)</td>
</tr>
</tbody>
</table>

From (1), \( A + B + 1 = A - \overline{B} \), and hence

\[ A + B = (A, B) - 1 \]  

(4)

Thus, by pairing up NB operands and by including a ‘-1’ in the error-correcting word at the corresponding position for each RB partial product, the sum of A and B is obtained. NB operands were expressed in one’s complement format, which requires an additional 1 to be added into the error-correcting word for every negative NB operand. The error-correcting word was of the form \( 0X0Y0X0Y \), where \( X \in \{0,1\} \) and \( Y \in \{0,1\} \). Both \( X \) and \( Y \) are functions of RB and Booth recoding terms.

Although the above method eliminated the carry propagate operation, it added an extra error-correction block into the partial product reduction tree. Also, the error-correction method described in this multiplier put restrictions in the number of bits that can be multiplied. For a 64-bit multiplier, there will be more than 16 RB partial products including the error-correction term. This will require the use of an extra stage of RBAs, thereby significantly increasing the multiplication time. In general, if the number of partial products were perfect powers of 2, the multiplier will be inefficient.

### III. Fast Partial Product Generator

The proposed partial product generator generates RB partial products, without any carry propagation delay or any additional hardware. For a multiplicand ‘y’ the radix-4 Booth encoder will have five different NB partial products \( \{-2y, y, 0, y, 2y\} \). Instead of generating \(-2y\) and \(-y\) in their complement form the multiplier retains the partial products in their one’s complement form and introduces an extra bit ‘1’ along with the partial products. The NB partial product \(-y\) obtained from Booth encoder is expressed as \((\overline{y},1)\), where \( y \) is the one’s complement of \( y \). The set of partial products obtained from Booth encoder is represented as \( \{2y, 1, (\overline{y},1), (0,0), (y,0), (2y,0)\} \).

A NB partial product \( A \) can be represented as

\[ A = (A + a) \]  

(5)
where \( A^* = \overline{A} \) and \( a = 1 \), when \( A \) is negative and \( A^* = A \) and \( a = 0 \), when \( A \) is positive.

The sum of two NB partial products \( A \) and \( B \) can now be expressed as

\[
A + B = (A^* + a) + (B^* + b)
\]

\[
= A^* + B^* + a + b
\]

\[
= (A^* - B^*) - 1 + a + b
\]

Using the RB Encoding 2 shown in Table I, the above equation can be expressed as

\[
A + B = (A^*, B^*) - 1 + a + b
\]

(6)

For different positive and negative numbers \( A \) and \( B \), the values of \( a \) and \( b \) will be chosen according to Table II. It can be observed that \( a \) and \( b \) are nothing but the sign bits of \( A \) and \( B \) respectively.

If \( Z = a + b - 1 \), Equation 6 can be modified as

\[
A + B = (A^*, B^*) + Z
\]

(7)

where \( Z \) can be coded according to Table II.

The extra RB digit from each RB operand forms an extra operand, which can be fed into the next partial product accumulation stage as shown in Makino [5]. This correction-word will be having the format \( \ldots 0Z000Z000Z \), where \( Z \in \{1, 0, -1\} \). The addition of two NB partial products \( A = -10 \) and \( B = -20 \) according to Table II encoding is shown in Fig. 1. The two partial products are grouped along with the extra bit. In this case both numbers are expressed in their one’s complement representations. The extra bits are ‘1’ for both, and are shown separately in Fig. 1. The bit pair \((A, B)\) is also shown in Fig 1. These bit pairs represent the sum \( A + B \) in RB notation. The equivalent RB number can be obtained using Encoding 2 in Table I and is shown at the bottom. The extra bit position is also assigned unit weight. The RB result obtained can be reconverted into its equivalent decimal value using a negative weight for the MSB bit. This results in the final sum of \(-30\).

**Figure 1.** Example of RBPPG using one’s complement arithmetic

The above method avoids any kind of carry propagate operation during partial product generation, and simply expresses the partial products in one’s complement NB format for a negative number. The extra bit for each NB partial product is same as the sign bit of each operand. Contrary to Kim’s technique [10], the correction bit \( Z \) is found directly from the grouping, instead of a combination of RB and Booth recoding terms. Also, the correction digit is limited to one per RB partial product when compared to one per NB partial product in earlier designs. The RBPPG does not use any gates (including inverters) for obtaining the corrected RB partial product.

The comparison of various PPG blocks for a 54x54-bit and 64x64-bit multiplier is shown in Table III. The number of partial products after the encoding is shown for each multiplier. Our PPG design exhibits the highest amount of reduction among 54x54-bit multipliers. The details regarding the number of partial products for 54-bit multipliers was given in [5, 8, 10], whereas those for 64-bit multipliers were computed by us. It may be noted that in the case of 64-bit multipliers all the earlier multiplier formats exceed the optimum number of partial products for a 4 stage partial product accumulator.

**Table III. Number of partial products in different multiplier formats**

<table>
<thead>
<tr>
<th>PPG Design</th>
<th>54x54 Reduction (%)</th>
<th>64x64 Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Besli [8]</td>
<td>18</td>
<td>33.3</td>
</tr>
<tr>
<td>Makino [5]</td>
<td>15</td>
<td>27.7</td>
</tr>
<tr>
<td>Kim [10]</td>
<td>15</td>
<td>27.7</td>
</tr>
<tr>
<td>Ours</td>
<td>14</td>
<td>25.9</td>
</tr>
</tbody>
</table>
IV. PARTIAL PRODUCT ACCUMULATION

Partial product accumulation of previous multipliers, Makino [5] and Kim [10] were similar in structure. Our correction-word can replace its counterpart in Kim [10], thereby reducing the effort to combine RB and Booth recoding terms. Another method of improving partial product accumulation would be to exploit the carry-free addition scheme of the RBA. The carry propagation in RBAs is limited to 2 RBA cells for Encoding 2, and 3 RBA cells for Encoding 1. Since the correction bits have three zeros in between them, they can be fed into the carry digit of an RBA array without causing any carry propagation. This will avoid the error-correcting operand required in previous multipliers. The block diagram of a 64-bit multiplier using the new technique is shown in Fig. 2. Fast RBAs defined in Jose et al. [11] can be used for RB addition using the alternate Encoding 2 in Table I.

Our design could also eliminate the error-correction block used in the partial product accumulator [10]. This will enable us to extend the number of bits in the multiplier to 64 bits, while keeping the number of adder stages to four. Similar methods can be followed in the design of any multiplier having the number of partial products as perfect powers of two (64x64, 32x32, 16x16, etc). This design will accommodate RB encoding in such multipliers while enjoying the benefits of both lesser number of partial products with optimum number of adder stages.

V. CONCLUSIONS

A new partial product generation technique for Booth multipliers has been proposed by eliminating the carry propagation delay encountered in generating the negative partial products in two’s complement form. This is achieved by expressing the partial products in one’s complement form together with an extra bit. This technique replaces the error-correcting word in earlier designs with one error digit per RB operand, which can be added along with the RBA tree. The multiplier width can now be extended to perfect powers of 2, without increasing the number of stages of RBAs in the partial product accumulation stage. The use of radix-4 Booth encoding combined with our technique results in 78% reduction in the number of partial products generated. The selection of the particular RB encoding also allows us to take advantage of a faster RBA cell, thereby speeding up multiplication from all fronts.

REFERENCES