HIGH PERFORMANCE RADIX-16 BOOTH PARTIAL PRODUCT GENERATOR FOR 64-BIT BINARY MULTIPLIERS

1CH.AMANI TULASI  2G.V.V.S.RAJESH M.Tech., Assistant Professor
1,2Eluru College of Engineering And Technology, Duggirala, Pedavegi West Godavari, Andhra Pradesh, INDIA

Abstract: Redundant Binary Partial Product Generator technique are used to reduce by one row the maximum height of the partial product array generated by a radix-16 Modified Booth Encoded multiplier, without any raise in the delay of the partial product creation Block. In this paper, we describe an optimization for binary radix-16 (modified) Booth recoded multipliers to reduce the maximum height of the partial product columns to \([n/4]\) for \(n = 64\)-bit unsigned operands. This is in contrast to the conventional maximum height of \([{(n + 1)}/4]\). Therefore, a reduction of one unit in the maximum height is achieved. This Arithmetic multipliers increase the performance of ALU and Processors. We evaluate the proposed approach by comparison with Normal Booth Multiplier. Logic synthesis showed its efficiency in terms of area, delay and power. Simulation results show that the proposed Multiplier based designs significantly improve the area, delay and power consumption when the word length of each operand in the multiplier is 64 & 128 bits. The proposed architecture of this paper analysis the delay and area using Xilinx 14.5.

Index Terms: Error correction codes, high-speed networking, memory, single error correction (SEC).

I. INTRODUCTION

Digital multipliers are widely used in arithmetic units of microprocessors, multimedia and digital signal processors. Many algorithms and architectures have been proposed to design high-speed and low power multipliers. A normal binary (NB) multiplication by digital circuits includes three steps. In the first step, partial products are generated; in the second step, all partial products are added by a partial product reduction tree until two partial product rows remain. In the third step, the two partial product rows are added by a fast carry propagation adder. Two methods have been used to perform the second step for the partial product reduction. A first method uses 4:2 compressors, while a second method uses redundant binary (RB) numbers. Both methods allow the partial product reduction tree to be reduced at a rate of 2:1.

The redundant binary number representation has been introduced by Avizienis to perform signed-digit arithmetic; the RB number has the capability to be represented in different ways.

For example, in radix-16 signed digit recoding the digit set is \([-8,-7,\ldots,0,\ldots,7,8]\), so that some odd multiples of the multiplicand have to be generated. Specifically, it is required to generate \(\times 3, \times 5,\) and \(\times 7\) multiples (\(\times 6\) is obtained by simple shift of \(\times 3\)). The generation of each of these odd multiples requires a two term addition or subtraction, yielding a total of three carry-propagate additions.

II. LITERATURE SURVEY

The conventional modified Booth encoding (MBE) generates an irregular partial product array because of the extra partial product bit at the least significant bit position of each partial product row. In this brief, a simple approach is proposed to generate a regular partial product array with fewer partial product rows and negligible overhead, thereby lowering the complexity of partial product reduction and reducing the area, delay, and power of MBE multipliers. The proposed approach can also be utilized to regularize the partial product array of post truncated MBE multipliers. Implementation results demonstrate that the proposed MBE multipliers with a regular partial product array really achieve significant improvement in area, delay, and power consumption when compared with conventional MBE multipliers.

Complex arithmetic operations are widely used in Digital Signal Processing (DSP) applications. In this work, we focus on optimizing the design of the fused Add-Multiply (FAM) operator for increasing performance. We investigate techniques to implement the direct recoding of the sum of two numbers in its Modified Booth (MB) form. We introduce a structured and efficient recoding technique and explore three different schemes by incorporating them in FAM
designs. Comparing them with the FAM designs which use existing recoding schemes, the proposed technique yields considerable reductions in terms of critical delay, hardware complexity and power consumption of the FAM unit.

EXISTING METHOD

The architecture of the basic radix-16 Booth multiplier is shown in Fig.1. For sake of simplicity, but without loss of generality, we consider unsigned operands with n = 64. Let us denote with X the multiplicand operand with bit components xi (i = 0 to n − 1, with the least-significant bit, LSB, at position 0) and with Y the multiplier operand and bit components yi.

The first step is the recoding of the multiplier operand: groups of four bits with relative values in the set {0, 1 . . . 14, 15} are recoded to digits in the set {−8, −7, . . . , 0, . . . , 7, 8} (minimally redundant radix-16 digit set to reduce the number of multiples). This recoding is done with the help of a transfer digit ti and an interim digit wi.

The recoded digit zi is the sum of the interim and transfer digits zi = wi + ti. When the value of the four bits, vi, is less than 8, the transfer digit is zero and the interim digit wi = vi. For values of vi greater than or equal to 8, vi is transformed into vi = 16 − (16 − vi), so that a transfer digit is generated to the next radix-16 digit position (ti+1) and an interim digit of value wi = −(16 − v) is left. That is

0 ≤ vi < 8 : ti+1 = 0 wi = vi wi ∈ [0, 7]

8 ≤ vi ≤ 15 : ti+1 = 1 wi = −(16 − vi)

wi ∈ [−8, −1].

The transfer digit corresponds to the most-significant bit (MSB) of the four-bit group, since this bit determines if the radix-16 digit is greater than or equal to 8. The final logical step is to add the interim digits and the transfer digits (0 or 1) from the radix-16 digit position to the right. Since the transfer digit is either 1 or 0, the addition of the interim digit and the transfer digit results in a final digit in the set {−8, −7, . . . , 0, . . . , 7, 8}. After recoding, the partial products are generated by digit multiplication of the recoded digits times the multiplicand X.

For the set of digits {−8, −7, . . . , 0, . . . , 7, 8}, the multiples 1X, 2X, 4X, and 8X are easy to compute, since they are obtained by simple logic shifts. The negative versions of these multiples are obtained by bit inversion and addition of a 1 in the corresponding position in the bit array of the partial products. The generations of 3X, 5X, and 7X (odd multiples) requires carry-propagate adders (the negative versions of these multiples are obtained as before). Finally, 6X is obtained by a simple one bit left shift of 3X.

After the generation of the partial product bit array, the reduction (multi operand addition) from a maximum height of 17 (for n = 64) to 2 is performed. The methods for multi operand addition are well known, with a common solution consisting of using 3 to 2 bit reduction with full adders (or 3:2 carry-save adders) or 4 to 2 bit reduction with 4:2 carry-save adders. The delay and design effort of this stage are highly dependent on the maximum height of the bit array. It is recognized that reduction arrays of 4:2 carry-save adders may lead to more regular layouts.

For instance, with a maximum height of 16, a total of 3 levels of 4:2 carry-save adders would be necessary. A maximum height of 17 leads to different approaches that may increase the delay and/or require to use arrays of 3:2 carry-save adders interconnected to minimize delay [20]. After the reduction to two operands, a carry-propagate addition is performed. This addition may take advantage of the specific signal arrival times from the partial product reduction step.

Fig. 1: Partial product generation.
III. PROPOSED METHOD

To reduce the maximum height of the partial product bit array we perform a short carry-propagate addition in parallel to the regular partial product generation. This short addition reduces the maximum height by one row and it is faster than the regular partial product generation. Fig. 2(b) shows the elements of the bit array to be added by the short adder. Fig. 2(c) shows the resulting partial product bit array after the short addition. Comparing both figures, we observe that the maximum height is reduced from 17 to 16 for n = 64.

Fig. 2: Radix-16 partial product reduction array.

Since the partial products are left-shifted four bit positions with respect to each other, a costly sign extension would be necessary. However, the sign extension is simplified by concatenation of some bits to each partial product (S is the sign bit of the partial product and C is S complemented): CSSS for the first partial product and 111C for the rest of partial products (except the partial product at the bottom that is non negative since the corresponding multiplier digit is 0 or 1). The bits denoted by b in Fig. 2 corresponds to the logic 1 that is added for the two's complement for negative partial products.

![Diagram](image1)

Fig. 4 shows the implementation of part A. The compound adder determines speculatively the two possible results. Once the carry-in is obtained (from part B), the correct result is selected by a multiplexer. Note that the compound adder is of only five bits, since the propagation of the carry through the most significant three ones is straightforward. The computation of part B is more complicated. The main issue is that we need the 7 least-significant bits of

Fig. 3: Detail of the elements to be added by the short addition.

We perform the computation in two concurrent parts A and B as indicated in Fig. 3. The elements of the part A are generated faster than the elements of part B. Specifically the elements of part A are obtained from:

1) the sign of the first partial product: this is directly obtained from bit y3 since there is no transfer digit from a previous radix-16 digit;

2) bits 3 to 7 of partial product 16: the recoded digit for partial product 16 can only be 0 or 1, since it is just a transfer digit. Therefore the bits of this partial product are generated by a simple AND operation of the bits of the multiplicand X and bit y63 (that generates the transfer from the previous digit).

Therefore, we decided to implement part A as a speculative addition, by computing two results, a result with carry-in = 0 and a result with carry-in = 1. This can be computed efficiently with a compound adder. Fig. 4 shows the implementation of part A. The compound adder determines speculatively the two possible results. Once the carry-in is obtained (from part B), the correct result is selected by a multiplexer.
partial product 15. Of course waiting for the generation of partial product 15 is not an option since we want to hide the short addition delay out of the critical path.

Fig. 4: Speculative addition of part A.

Fig. 5: Computation of part B.

Fig. 6 shows the recoding and partial product generation stage including the high level view of the hardware scheme proposed.

The way we compute part B may still lead to an inconsistency with the computation of the most significant part of partial product 15. Specifically, when partial product 15 is the result of an odd multiple, a possible carry from the 7 least-significant bits is already incorporated in the most significant part of the partial product. During the computation of part B we should not produce again this carry.

This issue is solved as follows. Let us consider first the case of positive odd multiples. Fig. 5 shows that the computation of part B may generate two carry outs: the first from the 3:2 carry-save adder (Cout1), and the second from the carry-propagate adder (Cout2). To avoid inconsistencies, we detect the carry propagated to the most significant part of the partial product 15 (we call this CM) and subtract it from the two carries generated in part B.

**EXTENSION METHOD**

This array height reduction method can also be implemented to radix-16 algorithm with 128-bit multiplication. After the generation of the partial product bit array, the reduction (multi operand addition) from a maximum height of 33 (for n = 128) to 2 is performed. Here we are going to generate 32 PP by reducing array height by one.
IV. RESULTS

Simulation:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTL Schematic:

Technology Schematic:

Design Summary:

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Timing Summary:

EXTENSION METHOD:

Simulation:

RTL schematic:
Technology schematic:

Design Summary:

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Sigs</td>
<td>152</td>
<td>96</td>
<td>100%</td>
</tr>
<tr>
<td>Number of Equits</td>
<td>515</td>
<td>130</td>
<td>75%</td>
</tr>
<tr>
<td>Number of Total</td>
<td>352</td>
<td>40</td>
<td>88%</td>
</tr>
<tr>
<td>Number of MAXRES</td>
<td>4</td>
<td>4</td>
<td>100%</td>
</tr>
</tbody>
</table>

Timing Summary:

Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 45.024ns

V. CONCLUSION

The multiplier using the proposed algorithm achieves better power-delay products than those achieved by conventional Booth multipliers. Here, we have presented a method to reduce by one the maximum height of the partial product array for 64-bit,128-bit radix-16 Booth recoded magnitude multipliers. This reduction may allow more flexibility in the design of the reduction tree of the pipelined multiplier and achieved with no extra delay for $n \geq 32$ for a cell-based design. We believe that the proposed Booth algorithm can be broadly utilized in general processors as well as digital signal processors, mobile application processors, and various arithmetic units that use Booth encoding.

REFERENCES