64-Bit Radix_16 Booth Multiplier Using Partial Products

M.BHARATH KUMAR, F.ANTHONY CHERATHAN, M.VAKHIL KUMAR, M.RAMA KRISHNA

Students, Lecturer
Department of ECE,
Andhra Loyola Institute Of Engineering And Technology, Vijayawada.

Abstract: 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 reduction may add flexibility during the design of the pipelined multiplier to meet the design goals; it may allow further optimizations of the partial product array reduction stage in terms of area/delay/power and/or may allow additional addends to be included in the partial product array without increasing the delay. The method can be extended to Booth recoded radix-8 multipliers, signed multipliers, combined signed/unsigned multipliers, and other values of \(n\).

Introduction

Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets

1. High speed,
2. Low power consumption,
3. Regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed,
4. Low power and compact VLSI implementation.

Multimedia and Digital Signal Processing (DSP) applications (e.g., Fast Fourier Transform (FFT), audio/video Co Decs) carry out a large number of multiplications with coefficients that do not change during the execution of the application. Since the multiplier is a basic component for implementing computationally intensive applications, its architecture seriously affects its performance. Constant coefficients can be encoded to contain the least non-zero digits using the Canonic Signed Digit (CSD) representation [1]. CSD multipliers comprise the fewest non-zero partial products, which in turn decreases their switching activity. However, the CSD encoding involves serious limitations. Folding technique [2], which reduces silicon area by time multiplexing many operations into single functional units, e.g., adders, multipliers, is not feasible as the CSD-based multipliers are hard-wired to specific coefficients. In [3], a CSD-based programmable multiplier design was proposed for groups of pre-determined coefficients that share certain features. The size of ROM used to store the groups of coefficients is significantly reduced as well as the area and power consumption of the circuit. However, this multiplier design lacks flexibility since the partial products generation unit is designed specifically for a group of coefficients and cannot be reused for another group. Also, this method cannot be easily extended to large groups of pre-determined coefficients attaining at the same time high efficiency. Modified Booth (MB) encoding [4]–[7] tackles the aforementioned limitations and reduces to half the number of partial products resulting in reduced area, critical delay and power consumption. However, a dedicated encoding circuit is required and the partial products generation is more complex. In [8], Kim et al. proposed a technique similar to [3], for designing efficient MB multipliers for groups of pre-determined coefficients with the same limitations described in the previous paragraph. In [9], [10], multipliers included in butterfly units of FFT processors use standard coefficients stored in ROMs. In audio [11], [12] and video [13], [14] Co Decs, fixed coefficients stored in memory, are used as multiplication inputs. Since the values of constant coefficients are known in advance, we encode the coefficients off-line based on the MB encoding and store the MB encoded coefficients (i.e., 3 bits per digit) into a ROM. Using this technique [15]–[17], the encoding circuit of the MB multiplier is omitted. We refer to this design as pre-encoded MB multiplier. Then, we explore a Non-Redundant radix-4 Signed Digit (NR4SD) encoding scheme extending the serial encoding techniques of [6], [18]. The proposed NR4SD encoding scheme uses one of the following sets of digit values: \([-1,0,+1,+2]\) or \([-2,-1,0,+1]\). In order to cover the dynamic range of the 2's complement form, all digits of the proposed representation are encoded according to NR4SD except the most significant one that is MB encoded. Using the proposed encoding formula, we pre-encode the standard coefficients and store them into a ROM in a condensed form (i.e., 2 bits per digit). Compared to the pre-encoded MB multiplier in which the encoded coefficients need 3 bits per digit, the proposed NR4SD scheme reduces the memory size. Also, compared to the MB form, which uses five digit values \([-2,-1,0,+1,+2]\), the proposed NR4SD encoding uses four digit values. Thus, the NR4SD-based pre-encoded multipliers include a less complex partial products generation circuit. We explore the efficiency of the aforementioned pre-encoded multipliers taking into account the size of the coefficients’ ROM.
MODIFIED BOOTH ALGORITHM

Modified Booth (MB) is a redundant radix-4 encoding technique [6], [7]. Considering the multiplication of the 2’s complement numbers A, B, each one consisting of n=2k bits, B can be represented in MB form as:

\[ B = hbn-1 \ldots b0i20s = -b2k-122k-1 + 2k-2 X i=0 \]

\[ bi2i \]

= h b MB k-1 ...b MB 0 i MB = k-1 X j=0

b MB j 22j.

(1)

Digits b MB j \( \in \{-2,-1,0,1,2\}, 0 \leq j \leq k-1 \), are formed as follows: b MB j = -b2j+1 + b2j + b2j-1, (2) where b-1 = 0. Each MB digit is represented by the bits s, one and two (Table 1). The bit s shows if the digit is negative (s=1) or positive (s=0). One shows if the absolute value of a digit equals 1 (one=1) or not (one=0). Two shows if the absolute value of a digit equals 2 (two=1) or not (two=0). Using these bits, we calculate the MB digits b MB j as follows: b MB j = (−1)sj ·(one j + 2twoj). (3) Equations (4) form the MB encoding signals.

**Table 1**

<table>
<thead>
<tr>
<th>b2j+1</th>
<th>b2j</th>
<th>b2j-1</th>
<th>b2MB</th>
<th>s2j</th>
<th>one2j</th>
<th>two2j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

MB digit is represented by the bits s, one and two (Table 1). The bit s shows if the digit is negative (s=1) or positive (s=0). One shows if the absolute value of a digit equals 1 (one=1) or not (one=0). Two shows if the absolute value of a digit equals 2 (two=1) or not (two=0). Using these bits, we calculate the MB digits b MB j as follows: b MB j = (−1)sj ·(one j + 2twoj). (3) Equations (4) form the MB encoding signals. sj = b2j+1, one j = b2j-1 ⊕ b2j, two j = (b2j+1 ⊕ b2j) ∧ one j. (4)

NON-REDUNDANT RADIX-4 SIGNED DIGIT ALGORITHM

In this section, we present the Non-Redundant radix-4 Signed-Digit (NR4SD) encoding technique. As in MB form, the number of partial products is reduced to half. When encoding the 2’s complement number B,

**NR4SD− Algorithm**

Step 1: Consider the initial values j = 0 and c0=0. Step 2: Calculate the carry c2j+1 and the sum n+ 2j of a Half Adder (HA) with inputs b2j and c2j (Fig. 1a). c2j+1 = b2j ∧ c2j, n+ 2j = b2j ⊕ c2j. Step 3: Calculate the positively signed carry c2j+2 (+) and the negatively signed sum n− 2j+1 (−) of a Half Adder* (HA*) with inputs b2j+1 (+) and c2j+1 (+) (Fig. 1a). The outputs c2j+2 and n− 2j+1 of the HA* relate to its inputs as follows: 2c2j+2 −n− 2j+1 = b2j+1 + c2j+1. The following Boolean equations summarize the HA* operation: c2j+2 = b2j+1 + c2j+1, n− 2j+1 = b2j+1 ⊕ c2j+1. Step 4 Calculate the value of the bNR− j digit. bNR− j = 2n− 2j+1 + n+ 2j. (5) Equation (5) results from the fact that n− 2j+1 is negatively signed and n+ 2j is positively signed. Step 5: j := j + 1. Step 6: If (j < k−1), go to Step 2. If (j = k−1), encode the most significant digit based on the

**NR4SD+ Algorithm**

Step 1: Consider the initial values j = 0 and c0=0. Step 2: Calculate the carry c2j+1 and the sum n+ 2j of a Half Adder (HA) with inputs b2j and c2j (Fig. 1a). c2j+1 = b2j ∧ c2j, n+ 2j = b2j ⊕ c2j. Step 3: Calculate the positively signed carry c2j+2 (+) and the negatively signed sum n− 2j+1 (−) of a Half Adder* (HA*) with inputs b2j+1 (+) and c2j+1 (+) (Fig. 1a). The outputs c2j+2 and n− 2j+1 of the HA* relate to its inputs as follows: 2c2j+2 −n− 2j+1 = b2j+1 + c2j+1. The following Boolean equations summarize the HA* operation: c2j+2 = b2j+1 + c2j+1, n− 2j+1 = b2j+1 ⊕ c2j+1. Step 4 Calculate the value of the bNR− j digit. bNR− j = 2n− 2j+1 + n+ 2j. (5) Equation (5) results from the fact that n− 2j+1 is negatively signed and n+ 2j is positively signed. Step 5: j := j + 1. Step 6: If (j < k−1), go to Step 2. If (j = k−1), encode the most significant digit based on the
MB algorithm and considering the three consecutive bits to be $b_{2k-1}$, $b_{2k-2}$ and $c_{2k-2}$ (Fig. 1b). If ($j = k$), stop. Table 2 shows how the NR4SD− digits are formed. Equations (6) show how the NR4SD− encoding signals $\text{one}_+ j$, $\text{one}_- j$ and $\text{two}_- j$ of Table 2 are generated.

$$\text{one}_+ j = n− 2j+1 \land n+ 2j, \text{one}_- j = n− 2j+1 \lor n+ 2j, \text{two}_- j = n− 2j+1 \land n+ 2j.$$  

(6)

The minimum and maximum limits of the dynamic range in the NR4SD− form are $-2n−1−2n−3−2n−5\ldots−2 < -2n−1$ and $2n−1 + 2n−4 + 2n−6 +\cdots+ 1 > 2n−1−1$. We observe that the NR4SD− form has larger dynamic range than the 2’s complement form.

### NR4SD+ Algorithm

Step 1: Consider the initial values $j = 0$ and $c0=0$. Step 2: Calculate the carry positively signed $c2j+1 (+)$ and the negatively signed sum $n− 2j$ of a HA* with inputs $b2j (+)$ and $c2j (+)$ (Fig. 2a). The carry $c2j+1$ and the sum $n− 2j$ of the HA* relate to its inputs as follows: $2c2j+1 − n−$. Step 3: Calculate the carry $c2j+2$ and the sum $n+ 2j+1$ of a HA with inputs $b2j$ and $c2j+1$. $c2j+2 = b2j \land c2j+1$, $n+ 2j+1 = b2j \oplus c2j+1$. Step 4: Calculate the value of the $bNR_+ j$ digit, $bNR_+ j = 2n+ 2j+1 − n− 2j$. (7) Equation (7) results from the fact that $n+ 2j+1$ is positively signed and $n− 2j$ is negatively signed. Step 5: $j := j + 1$. Step 6: If ($j < k−1$), go to Step 2. If ($j = k−1$), encode the most significant digit according to MB algorithm and considering the three consecutive bits to be $b2k−1$, $b2k−2$ and $c2k−2$ (Fig. 2b). If ($j = k$), stop. Table 3 shows how the NR4SD+ digits are formed. Equations (8) show how the NR4SD+ encoding signals $\text{one}_+ j$, $\text{one}_- j$ and $\text{two}_+ j$ of Table 4 are generated.

$$\text{one}_+ j = n+ 2j+1 \land n− 2j, \text{one}_- j = n+ 2j+1 \land n− 2j, \text{two}_+ j = n+ 2j+1 \land n− 2j.$$  

(8)

The minimum and maximum limits of the dynamic range in the NR4SD+ form are $-2n−1−2n−4−2n−6\ldots−1 < -2n−1$ and $2n−1+2n−3+2n−5\ldots+2 > 2n−1−1$. As observed in the NR4SD− encoding technique, the NR4SD+ form has larger dynamic range than the 2’s complement form. Considering the 8-bit 2’s complement number $N$, Table 4 exposes the limit values $-28 = -128$, $28−1 = 127$, and two typical values of $N$, and presents the MB, NR4SD− and NR4SD+ digits that result when applying the corresponding encoding techniques to each value of $N$ we considered. We added a bar above the negatively signed digits in order to distinguish.

### Table 2

<table>
<thead>
<tr>
<th>$N$'s Complement</th>
<th>NR4SD− Encoding</th>
<th>Digit</th>
<th>NR4SD+ Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001000</td>
<td>10011010</td>
<td>10011010</td>
<td></td>
</tr>
<tr>
<td>10110110</td>
<td>01011101</td>
<td>01011111</td>
<td></td>
</tr>
<tr>
<td>01111111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3

Numerical Examples of the Encoding Techniques

<table>
<thead>
<tr>
<th>$N$'s Complement</th>
<th>Modified Booth</th>
<th>2’s Complement</th>
<th>2’s Complement</th>
<th>2’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>001011102</td>
<td>001011102</td>
<td>001011102</td>
<td>001011102</td>
<td>001011102</td>
</tr>
<tr>
<td>010110102</td>
<td>010110102</td>
<td>010110102</td>
<td>010110102</td>
<td>010110102</td>
</tr>
<tr>
<td>011011102</td>
<td>011011102</td>
<td>011011102</td>
<td>011011102</td>
<td>011011102</td>
</tr>
<tr>
<td>01111111</td>
<td></td>
<td>01111111</td>
<td>01111111</td>
<td>01111111</td>
</tr>
</tbody>
</table>

The minimum and maximum limits of the dynamic range in the NR4SD+ form are $-2n−1−2n−4−2n−6\ldots−1 < -2n−1$ and $2n−1+2n−3+2n−5\ldots+2 > 2n−1−1$. As observed in the NR4SD− encoding technique, the NR4SD+ form has larger dynamic range than the 2’s complement form. Considering the 8-bit 2’s complement number $N$, Table 4 exposes the limit values $-28 = -128$, $28−1 = 127$, and two typical values of $N$, and presents the MB, NR4SD− and NR4SD+ digits that result when applying the corresponding encoding techniques to each value of $N$ we considered. We added a bar above the negatively signed digits in order to distinguish.
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. 3 shows the specific elements of the bit array (boxes) to be added by the short carry-propagate addition. In this figure, pi,j corresponds to the bit j of partial product i, s0 is the sign bit of partial product 0, c0 = NOT(s0), bi is the bit for the two’s complement of partial product i, and z is the ith bit of the result of the short addition. The selection of these specific bits to be added is justified by the fact that, in this way, the short addition delay is hidden from the critical path that corresponds to a regular partial product generation (this will be shown in Section IV). 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: • 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; • 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 [7]. 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 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. We decided to implement a specific circuit to overide the computation of the least-significant

In this section we evaluate the proposed method. The main goal of this section is to demonstrate that with current technologies, it is possible to “hide” the delay of the additional logic placed in parallel to the partial product generation, so that it is out of the critical path. First, we show the results of the hardware synthesis using state of the art CAD tools (Synopsys Design Compiler [21]). Second, we evaluate the impact of the proposed method on the whole multiplier for different pipeline choices. Then, we show a technology independent path analysis using a high level area delay model to have more insight in the component delays of the critical path. Synthesis With CAD Tools We have performed a hardware synthesis using Synopsys Design Compiler [21] with the STM 90nm CMOS standard cell library. For this library the delay of a FO4 is 45 ps (FO4 is the delay of an inverter of minimum size with a load of four inverters), and the area of a two-input NAND gate is 4.4 μm². We synthesized the full partial product generation stage for the basic scheme allowing Synopsys’ Design Ware [21] to choose the adder and the proposed scheme with hand coding of adders (we need the internal carry of the adders, so we were not able to use Design Ware in this case). We did not optimize the 3X adder as described for instance in [12], [22] and [23], since this optimization can not be applied to the 5X and 7X adders, so that the critical path remains the same. Fig. 7 shows the latency-area space for the two synthesized designs. For higher latency points, as expected, the proposed design has a slight increase in area. The fastest design point is roughly the same for the two designs, although the proposed design has a penalty of about 2 K additional NAND-2 gates with respect to the basic scheme. For the fastest design point, the cost of the additional hardware in the proposed scheme is about 500 NAND-2 gates (even less since 7 least-significant bits of one radix-16 regular partial product are not required), less than 1.8% of the hardware complexity of the partial product generation stage. Therefore, the extra 1.5 K NAND-2 gates corresponds to the penalty of not using Design Ware adders in the proposed design. Our synthesis experiment shows that the proposed scheme does not introduce any significant variation in the latency-area space of the partial product generation stage, confirming our hypothesis that the introduced hardware has a minor cost and is hidden from the critical path. Therefore, we have the benefit of reducing the maximum height of the partial product array by one unit without introducing any significant penalties in the partial product generation stage.
B. Impact on the Multiplier In the previous subsection, we provided the detail of the synthesis of the partial product generation with the proposed method. In this subsection, we evaluate the impact of our method on the whole multiplier. We implement a multiplier by the proposed method to reduce the partial products by one, and we compare its performance (maximum clock frequency, area and power dissipation) to a multiplier, referred as basic, with the standard partial product generation and an extra operand in the accumulation tree. A practical design of a 64 × 64 multiplier is normally pipelined to guarantee high-throughput. However, the placement of pipeline registers depends mostly on the specific technology and may vary from design to design. High radix multipliers are chosen because the shallower trees allow a significant power reduction, since the glitching power is limited to a few levels of gates in the tree. For this reason, it is realistic to place pipeline registers before the tree, i.e., store the partial products in the pipeline registers. Consequently, we evaluate two schemes: 1) a 2-stage pipelined design [see Fig. 8(a)] with pipeline register placed between the partial products generation (stage abbreviated as PPGEN in the figures and tables) and the tree (TREE); 2) a 3-stage design [see Fig. 8(b)] with an additional pipeline register placed between the tree and the final carry propagate adder (CPA). Other pipeline placements are not convenient because they will result in placing flip-flops inside functional units, such as CPAs or adder trees.

Synthesis with CAD Tools We have performed a hardware synthesis using Synopsys Design Compiler[21] with the STM 90nm CMOS standard cell library. For this library the delay of a FO4 is 45 ps (FO4 is the delay of an inverter of minimum size with a load of four inverters), and the area of a two-input NAND gate is 4.4 μm². We synthesized the full partial product generation stage for the basic scheme allowing Synopsys’ Design Ware [21] to choose the adder, and the proposed scheme with hand coding of adders (we need the internal carry of the adders, so we were not able to use Design Ware in this case). We did not optimize the 3X adder as described for instance in[12],[22] and[23], since this optimization can not be applied to the 5X and 7X adders, so that the critical path remains the same. Fig. 7 shows the latency-area space for the two synthesized designs. For higher latency points, as expected, the proposed design has a slight increase in area. The fastest design point is roughly the same for the two designs, although the proposed design has a penalty of about 2 K additional NAND-2 gates with respect to the basic scheme. For the fastest design point, the cost of the additional hardware in the proposed scheme is about 500 NAND-2 gates (even less since 7 least-significant bits of one radix-16 regular partial product are not required), less than 1.8% of the hardware complexity of the partial product generation stage. Therefore, the extra 1.5 K NAND-2 gates corresponds to the penalty of not using Design Ware adders in the proposed design.

Comparison:

<table>
<thead>
<tr>
<th>TYPE OF MULTIPLIER</th>
<th>AREA</th>
<th>MEMORY</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL 32-BIT MULTIPLIER</td>
<td>7%</td>
<td>255 MB</td>
<td>10.15 ns</td>
</tr>
<tr>
<td>32-BIT BOOTH MULTIPLIER</td>
<td>4%</td>
<td>321 MB</td>
<td>10.14 ns</td>
</tr>
<tr>
<td>32-BIT BOOTH MULTIPLIER (32 RADIX-16)</td>
<td>4%</td>
<td>301 MB</td>
<td>10.71 ns</td>
</tr>
</tbody>
</table>

Conclusion: Pipelined large word length digital multipliers are difficult to design under the constraints of core cycle time (for nominal voltage), pipeline depth, power and energy consumption and area. Low level optimizations might be required to meet these constraints. In this work, we have presented a method to reduce by one the maximum height of the partial product array for 64-bit radix-16 Booth recoded magnitude multipliers. This reduction may allow more flexibility in the design of the reduction tree of the pipelined multiplier. We have shown that this reduction is achieved with no extra delay for n ≥32foracell-based design. The method can be extended to Booth recoded radix-8 multipliers, signed multipliers and combined signed/unsigned multipliers. Radix-8 and radix-16 Booth recoded multipliers are attractive for low power designs, mainly to the lower complexity and depth of the reduction tree, and therefore they might be very popular in this era of power-constrained designs with increasing over heads due to wiring.

References:


