OPTIMIZED 64-BIT RADIX-16 BOOTH MULTIPLIER WITH ENHANCED PARTIAL PRODUCT REDUCTION FOR HIGH-PERFORMANCE COMPUTING

Kailash Sinha[1], P. Venkatesh[1], Sk Ahmed Pasha[1]

[1]Department of Electronics and Communication Engineering

[1]Sree Dattha Institute of Engineering and Science,Sheriguda, Hyderabad, Telangana

Abstract:

The Redundant Binary Partial Product Generator technique optimizes radix-16 Modified Booth Encoded multipliers by reducing the maximum height of the partial product array by one row without increasing the delay of the partial product creation block. This paper presents an improved binary radix-16 Booth recoded multiplier, minimizing the maximum height of partial product columns to [n/4] for 64-bit unsigned operands, compared to the conventional [(n+1)/4], achieving a one-unit reduction. This optimization enhances arithmetic multipliers, thereby improving the performance of ALUs and processors. The proposed design is evaluated against the standard Booth multiplier through logic synthesis, demonstrating superior efficiency in terms of area, delay, and power consumption. Simulation results confirm significant improvements in these parameters when the operand word length is 64 and n-bits. The reduction in partial product height leads to lower hardware complexity and improved computational speed. Additionally, the architecture is analyzed using Xilinx 14.2, ensuring a detailed assessment of delay and area. By streamlining partial product generation, the proposed technique contributes to more efficient high-speed multipliers, which are crucial for modern computing applications. The study emphasizes the effectiveness of the optimized design in reducing resource utilization while maintaining performance, making it a viable solution for power-efficient and high-performance digital signal processing and computing systems.

Keywords: Modified Booth Encoding, Radix-16, Pipeline, Multiplier, Enhanced, Carry Select Adder, Binary Excess Converter.

1. Introduction

Multipliers are fundamental hardware components in various digital and high-performance systems, including FIR filters, digital signal processors, and microprocessors. The demand for high-speed and low-power computing has driven extensive research into efficient multiplication algorithms and architectures. Numerous techniques have been developed to enhance multiplication performance, focusing on reducing power consumption, improving speed, and optimizing resource utilization. Technological advancements have enabled the design of multipliers that not only achieve high-speed operations but also ensure

regularity in their layout, making them highly suitable for VLSI implementation. Optimized multipliers contribute significantly to improving overall system efficiency, particularly in applications requiring real-time processing and complex arithmetic computations. Various architectures, such as Booth encoding, Wallace tree, and array multipliers, have been explored to achieve an optimal balance between power, delay, and area. Modern multiplication techniques leverage parallel processing and pipelining to enhance computation speed while minimizing energy consumption. Digital signal processing requires efficient multiplication operations with the highest possible speed without compromising the power budget. In general, basic multiplication algorithm can be divided into three following steps.1) partial product (pp) generation, 2) partial product reduction and 3) final carry propagated addition [1-2]. In the first step, a set of partial product rows is generated where each one is the result of the product of one bit of the multiplier by multiplicand. For example, if we consider the multiplication X x Y with both X and Y on n bits and of the form xn_1 . . . x0 and yn_1 . . . y0, then the ith row is, in general, a proper left shifting of yi x X, i.e., either a string of all zeros when yi = 0, or the multiplicand X itself when yi = 1. In this case, the number of PP rows generated during the first phase is clearly n [1-4]. Recoding of binary numbers was first hinted at by Booth [5] four decades ago. MacSorley [6] proposed a modification of Booth's algorithm a decade after. Modified booth encoding (MBE) [6] is a technique that has been introduced to reduce the no of pp rows with a maximum height of [n/2] +1 rows. More specifically, Two's complement multiplier [7] using radix-4 MBE generates a pp array with a maximum height of [n/2] rows without any increase of delay, each row of the pp array follows the one of the following possible values: all zeros, +X, +2X [8].This pp reduction may increases the speed of the multiplier. During pp reduction phase, all pp rows are reduced by using compression tree [9-10].Since the intermediate addition values is not important, the outcome of this phase is a result represented in redundant carry save form, i.e., as two rows, which allows for much faster implementations. The final (carry-propagated) addition has the task of adding these two rows and of presenting the final result in a non-redundant form, i.e., as a single row. In this work, we introduce an idea to reduce the pp array with a maximum height of [n/3] rows by using radix-8 booth recoding process. A similar study aimed at the reduction of the maximum height to [n/3] but using a different approach has recently presented interesting results in [11]. Thus, in the following, we will evaluate and compare the proposed approach with the technique in [7].

2. 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.

3. 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 radix16 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

$0 \leq vi < 8 : ti+1 = 0 \; wi = vi \; wi \in [0, 7]$

$8 \leq vi \leq 15 : ti+1 = 1 \; wi = -(16 - vi)$

$wi \in [-8,-1]$.

The transfer digit corresponds to the mostsignificant 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 generation 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.

III. PROPOSED METHOD

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. 1 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.

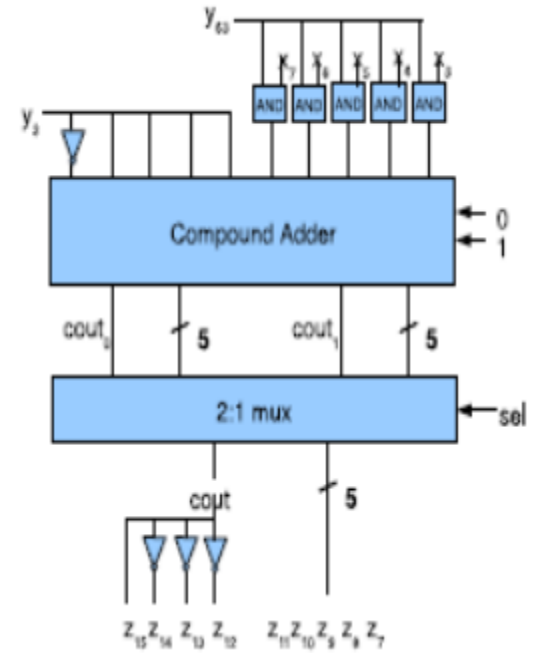

Fig. 1: Speculative addition of part A.

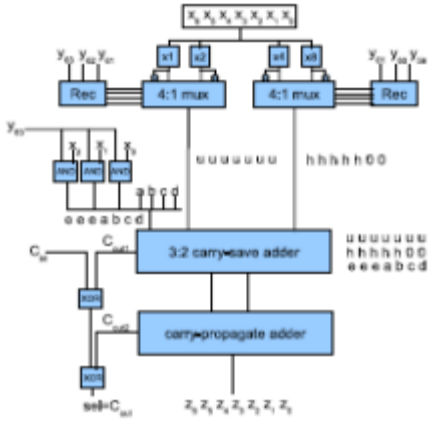

Fig. 2: Computation of part B.

Fig. 3 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. 3 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.

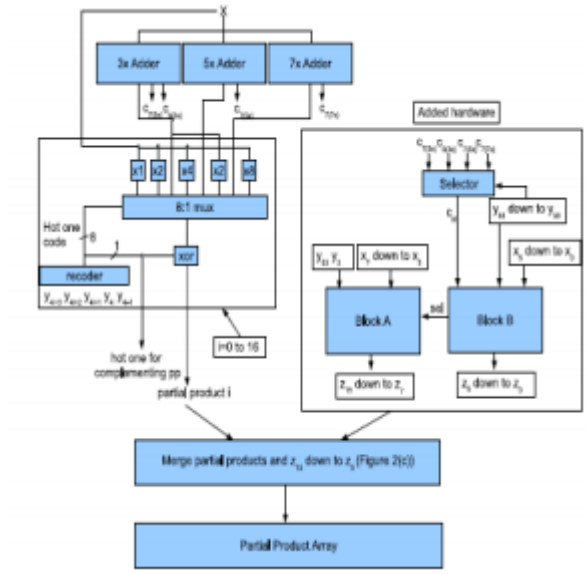

Fig. 3: High level view of the recoding and partial product generation stage including our proposed scheme.

This array height reduction method can also be implemented to radix-64 algo128-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.

4. RESULTS

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	21	4656	0%
Number of 4 input LUTs	37	9312	0%
Number of bonded IOBs	16	232	6%

Figure 4:- Design summary

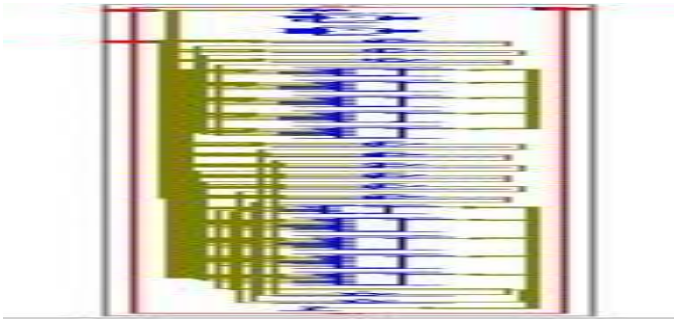

Figure 5:- RTL schematic

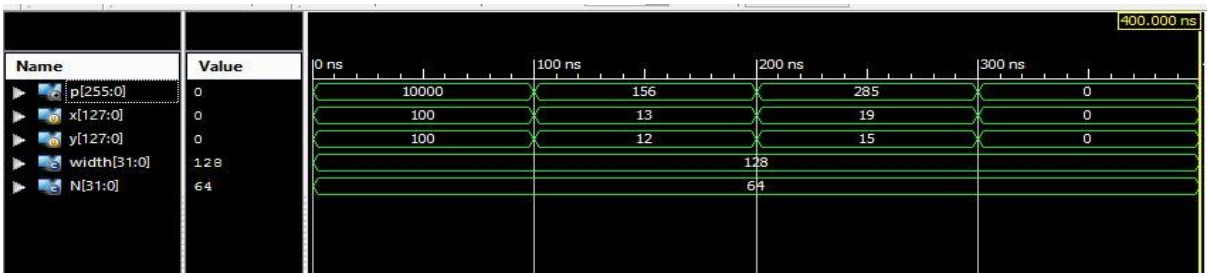

Figure 6:- Simulation results

```
Source:          x<0> (PAD)
Destination:     p<7> (PAD)

Data Path: x<0> to p<7>
                            Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   IBUF:I->O            13   1.106   0.905  x_0_IBUF (Madd_inv_x_lut<0>)
   LUT4:I1->O            3   0.612   0.520  Madd_inv_x_xor<3>11 (inv_x<3>)
   LUT3:I1->O            1   0.612   0.000  spp_0_mux0000<3>1 (spp_0_mux0000<3>1)
   MUXF5:I1->O           3   0.278   0.603  spp_0_mux0000<3>_f5 (spp_0_mux0000<3>)
   LUT4:I0->O            1   0.612   0.509  Madd_prod_cy<4>1_SW0 (N30)
   LUT4:I0->O            3   0.612   0.520  Madd_prod_cy<4>1 (Madd_prod_cy<4>)
   LUT2:I1->O            1   0.612   0.357  Madd_prod_xor<5>11 (p_5_OBUF)
   OBUF:I->O                 3.169          p_5_OBUF (p<5>)
  ----------------------------------------
  Total                     11.028ns (7.613ns logic, 3.415ns route)
                                     (69.0% logic, 31.0% route)
```

Figure 7:- Time Summary

5. 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

[1] S. Kuang, J. Wang, and C. Guo, "Modified booth multipliers with a regular partial product array," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 56, no. 5, pp. 404–408, May 2009.

[2] F. Lamberti et al., "Reducing the computation time in (short bit-width) twos complement multipliers," IEEE Trans. Comput., vol. 60, no. 2, pp. 148–156, Feb. 2011.

[3] N. Petra et al., "Design of fixed-width multipliers with linear compensation function," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, no. 5, pp. 947– 960, May 2011.

[4] S. Galal et al., "FPU generator for design space exploration," in Proc. 21st IEEE Symp. Comput. Arithmetic (ARITH), Apr. 2013, pp. 25–34.

[5] K. Tsoumanis et al., "An optimized modified booth recoder for efficient design of the add-multiply operator," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 4, pp. 1133–1143, Apr. 2014.

[6] A. Cilardo et al., "High speed speculative multipliers based on speculative carry-save tree," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 12, pp. 3426–3435, Dec. 2014. [7] M. Ercegovac and