# Partial Product Array Height Reduction Based High Performance 64-bit Radix-16 Booth Multiplier

**D. Ramadevi(1) & D. Raghava Reddy(2) & P. Malleswari (3)**

[1]PG Scholar, Dept of ECE, Narsaraopeta Institute of Technology, Yellamanda, Narsaraopeta, Guntur, Andhra Pradesh, India.

[2]HOD, Dept of ECE, Narsaraopeta Institute of Technology, Yellamanda, Narsaraopeta, Guntur, Andhra Pradesh, India.

[3]Assistant Professor, Dept of ECE, Narsaraopeta Institute of Technology, Yellamanda, Narsaraopeta, Guntur, Andhra Pradesh, India.

Received: May 23, 2018 Accepted: July 11, 2018

**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.*

***Keywords:*** *Binary multipliers, modified Booth recoding, radix-16.*

## I.INTRODUCTION

High speed digital multipliers are fundamental elements in signal processing and arithmetic based systems. The higher bit widths required of modern multipliers provide the opportunity to explore new architectures which would be impractical for smaller bit width multiplication.

Current implementations of binary multiplication follow the steps of [7]: 1) recoding of the multiplier in digits in a certain number system; 2) digit multiplication of each digit by the multiplicand, resulting in a certain number of partial products; 3) reduction of the partial product array to two operands using multioperand addition techniques; and 4) carry-propagate addition of the two operands to obtain the final result.

The recoding type is a key issue, since it determines the number of partial products. The usual recoding process recodes a binary operand into a signed-digit operand with digits in a minimally redundant digit set [7], [8]. Specifically, for radix-r ($r = 2^m$), the binary operand is composed of nonredundant radix-r digits (by just making groups of m bits), and these are recoded from the set $\{0, 1. . . r - 1\}$ to theset $\{-r/2, . . . ,-1, 0, 1, . . ., r/2\}$ to reduce the complexity of digit multiplications. For n-bit operands, a total of [n/m] partial products are generated for two's complement representation, and [(n + 1)/m] for unsigned representation.

In order to perform high speed multiplication, an encoding scheme, such as that proposed by Booth, is often used. The objective of Booth encoding is to reduce the number of partial products which are summed to generate the complete product, and thereby decrease the time required to compute the final product.

Radix-4 modified Booth is a widely used recoding method that recodes a binary operand into radix-4 signed digits in the set {−2, −1, 0, 1, 2}. This is a popular recoding since the digit multiplication step to generate the partial products only requires simple shifts and complementation. The resulting number of partial products is about n/2.

Higher radix signed recoding is less popular because the generation of the partial products requires odd multiples of the multiplicand which cannot be achieved by means of simple shifts, but require carry-propagate additions. For instance, for radix-16 signed digit recoding [9] the digit set is {−8,−7, . . . , 0, . . . , 7, 8}, so that some odd multiples of the multiplicand have to be generated. Specifically, it is required to generate ×3, ×5, and ×7 multiples (×6 is obtained by simple shift of ×3). The generation of each of these odd multiplies requires a two term addition or subtraction, yielding a total of three carry-propagate additions.

However, the advantage of the high radix is that the number of partial products is further reduced. For instance, for radix-16 and n-bit

operands, about n/4 partial products are generated. Although less popular than radix-4, there exist industrial instances of radix-8 [10]–[16] and radix-16 multipliers [17] in microprocessors implementations. The choice of these radices is related to area/delay/power optimization of pipelined multipliers (or fused multiplier adder as in the case of Intel Itanium microprocessor [17]), for balancing delay between stages and/or reduce the number of pipelining flip-flops.

A further consideration is that carry-propagate adders are today highly energy-delay optimized, while partial product reductions trees suffer the increasingly serious problems related to a complex wiring and glitching due to unbalanced signal paths. It is recognized in the literature that a radix-8 recoding leads to lower power multipliers compared to radix-4 recoding at the cost of higher latency (as a combinational block, without considering pipelining) [4], [18].

Moreover, although the radix-16 multiplier requires the generation of more odd multiples and has a more complex wiring for the generation of partial products [4], a recent microprocessor design [17] considered it to be the best choice for low power (under the specific constraints for this microprocessor).

In [1] and [2], some optimizations for radix-4 two's complement multipliers were introduced. Although for n-bit operands, a total of [n/2] partial products are generated, the resultingmaximum height of the partial product array is [n/2] + 1 elements to be added (in just one of the columns). This extra height by a single-bit row is due to the +1 introduced in the bit array to make the two's complement of the most significant partial product (when the recoded most significant digit of the multiplier is negative). The maximum column height may determine the delay and complexity of the reduction tree [7], [16].

In [1] and [2], authors showed that this extra column of one bit could be assimilated (with just a simplified three bit addition) with the most significant part of the first partial product without increasing the critical path of the recoding and partial product generation stage. The result is that the partial product array has a maximum height of [n/2]. This reduction of one bit in the maximum height might be of interest for high-performance short-bitwidth two's complement multipliers (small n) with tight cycle time constraints that are very common in SIMD digital signal processing applications.

Moreover, if n is a power of two, the optimization allows to use only 4-2 carry-save adders for the reduction tree, potentially leading to regular layouts [16]. These kind of optimizations can become particularly important as they may add flexibility to the "optimal" design of the pipelined multiplier. Optimal pipelining in fact, is a key issue in current and future multiplier (or multiplier-add) units: 1) the latency of the pipelined unit is very important, even for throughput oriented applications, as it impacts the energy consumption of the whole core [19]; and 2) the placement of the pipelining flip-flops should at the same time minimize total power, due to the number of flip-flops required and the unbalanced signal propagation paths.

Unsigned multiplication may produce a positive carry out during recoding (this depends of the value of n and the radix used for recoding), leading to one additional row, increasing the maximum height of the partial product array by one row, not just in one but in several columns. For all these reasons, we need to extend the techniques presented in [1] and [2]. In this work, we present a technique that allows partial product arrays of maximum height of [n/m] (with the goal of not increasing the delay of the partial product generation stage), for r > 4 and unsignedmultipliers. Since for the standard unsigned multiplier the maximum height is [(n + 1)/m], the proposed method allows a reduction of one row when n is a multiple of m.

Our technique is general, but its impact (reduction of one row without increasing the critical path of the partial product generation stage) depends on the specific timing of the different components. Therefore, we cannot claim a successful result for all practical values of r and n and different implementation technologies. Thus, we concentrate on a specific instance: a 64-bit radix-16 Booth recoded unsigned multiplier implemented with a synthesis tool and a standard-cell library. Weuse radix-16 since it is the most complex case, among the practical values of the radix, for the design of our scheme.

The unsigned multiplier is also more complex for the design of our scheme than the signed multiplier. We use 64 bits, since it is a representative large wordlength. The method proposed can be adapted easily to other instances (signed, combined unsigned/signed, radix-8 recoding, different values of n).

## II. BASIC RADIX-16 BOOTH MULTIPLIER

In this section, we describe briefly the architecture of the basic radix-16 Booth

multiplier (see [17] for instance). 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 [8]: 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 [7]. The recoded digit zi is the sum of the interim and transfer digits

$$z_i = w_i + t_i.$$

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

$$0 \le v_i < 8 : t_{i+1} = 0 \quad w_i = v_i \quad w_i \in [0, 7]$$

$$8 \le v_i \le 15 : t_{i+1} = 1 \quad w_i = -(16 - v_i) \quad w_i \in [-8, -1].$$

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}. Due to a possible transfer digit from the most significant radix-16 digit, the number of resultant radix-16 recoded digits is (n + 1)/4. Therefore, for n = 64 the number of recoded digits (and the number of partial products) is 17.

Note that the most significant digit is 0 or 1 because it is in fact just a transfer digit. 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.

Fig. 1 illustrates a possible implementation of the partial product generation. Five bits of the multiplier Y are used to obtain the recoded digit (four bits of one digit and one bit of the previous digit to determine the transfer digit to be added). The resultant digit is obtained as a one-hot code to directly drive a 8 to 1 multiplexer with an implicit zero output (output equal to zero when all the control signals of the multiplexer are zero). The recoding requires the implementation of simple logic equations that are not in the critical path due to the generation in parallel of the odd multiples (carry-propagate addition).

The XOR at the output of the multiplexer is for bit complementation (part of the computation of the two's complement when the multiplier digit is negative). Fig. 2(a) illustrates part of the resultant bit array for n = 64 after the simplification of the sign extension [7].

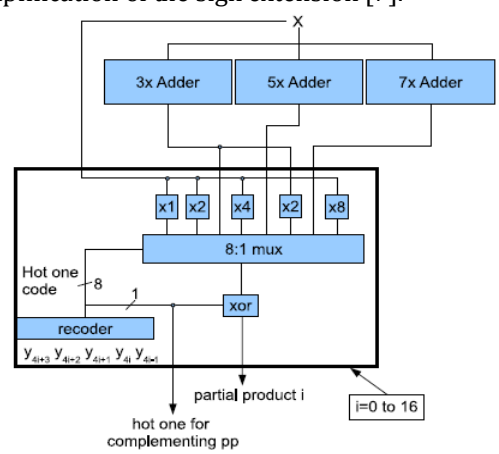



**Fig. 1. Partial product generation.**

In general, each partial product has n + 4 bits including the sign in two's complement representation. The extra four bits are required to host a digit multiplication by up to 8 and a sign bit due to the possible multiplication by negative multiplier digits. 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. 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 [16].

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.

## 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. 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 zi 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 below). We perform the computation in two concurrent parts A and B as indicated in Fig. 3.

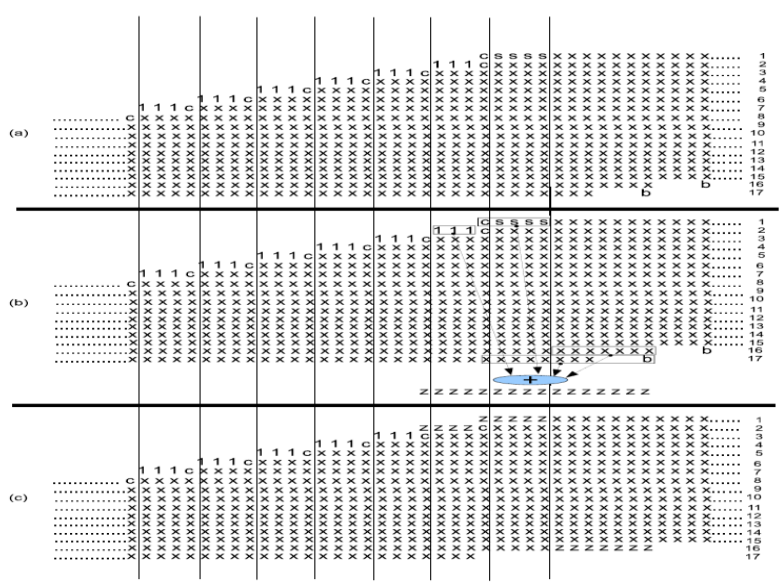



**Fig. 2. Radix-16 partial product reduction array.**

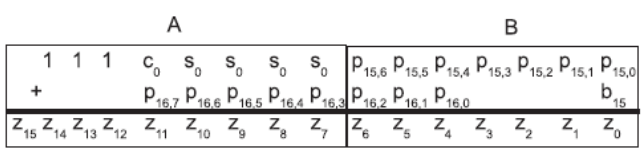



**Fig. 3. Detail of the elements to be added by the short addition.**

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.

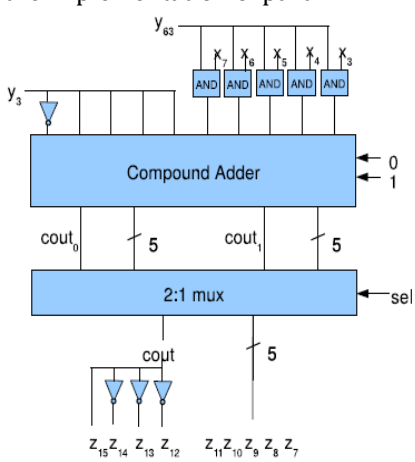



**Fig. 4. Speculative addition of part A.**

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 embed the computation of the least-significant bits of partial product 15 in the computation of part B (and also the addition of the bit b15). Note that for the method to be correct the computation of the partial product embedded in part B should be consistent with the regular computation performed for the most significant bits of partial product 15.

Fig. 5 shows the computation of part B. We decided to compute part B as a three operand addition with a 3:2 carry save adder and a carry-propagate adder. Two of the operands correspond to the least-significant bits of the partial product 15 and the other operand corresponds to the three least-significant bits of partial product 16 (that are easily obtained by an AND operation). We perform the computation of the bits of the radix-16 partial product 15 as the addition of two radix-4 partial products.

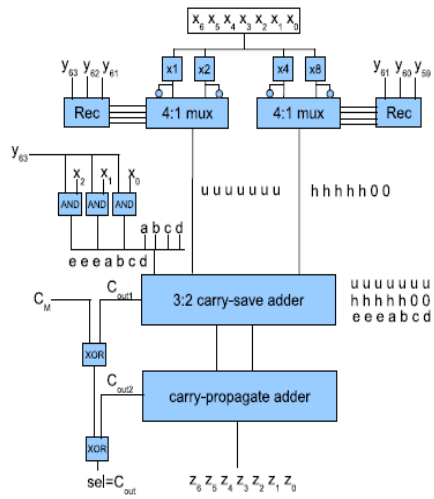



**Fig. 5. Computation of part B.**

Therefore, we perform two concurrent radix-4 recodings and multiple selection. The multiples of the least significant radix-4 digit are {−2, −1, 0, 1, 2}, while the multiples for the most significant radix-4 digit are {−8, −4, 0, 4, 8} (radix-4 digit set {−2, −1, 0, 1, 2}, but with relative weight of 4 with respect to the least-significant recoding). These two radix-4 recodings produce exactly the same digit as a direct radix-16 recoding for most of the bit combinations.

However, among the 32 5-bit combinations for a full radix-16 digit recoding, there are six not consistent with the two concurrent radix-4 recodings. Specifically:

- The bit strings 00100 and 11011 are recoded in radix-16 to 2 and −2 respectively. However, when performing two parallel radix-4 recodings the resulting digits are (4, −2) and (−4, 2) respectively. That is, the radix-4 recoding performs the computation of 2X (-2X) as 4X-2X (−4X + 2X). To have a consistent computation we modified the radix-4 recoders so that these strings produce radix-4 digits of the form (0, 2) and (0, −2).
- The bit strings 00101 and 00110 are recoded in radix-16 to 3 in both cases.

However, the resulting radix-4 digits are (4, −1). This means that the radix-4 recoding performs the computation of 3X as 4X-X. To address this inconsistency problem, in this case, we decided to implement the radix-16 multiple 3X as 4X-X. This avoids the combination of radix-4 digits (2, 1) and simplifies the multiplexers in Fig. 5.

- The bit strings 11001 and 11010 are recoded in radix-16 to −3 in both cases. However, the resulting radix-4 digits are (−4, 1). Therefore, for consistency, we proceed as in the previous case by generating the radix-16 multiple −3X as −4X + X.

To handle negative multiples, we select complemented inputs in the multiplexers and place 1 in a slot of the input of the 3:2 carry-save adder with relative binary weight equal to the absolute value of the corresponding radix-4 digit. These hot ones for two's complement are indicated in Fig. 5 as the string "abcd." For instance, if the least-significant radix-4 digit is −2 and the most significant radix-4 digit is −4, then c = 1 and b = 1. Therefore, "abcd" signals are obtained directly from the selection bits of the 4:1 multiplexers.

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.

Specifically, Table I shows the truth table to generate the carry out of part B. This truth table corresponds to the XOR of the three inputs. The CM carry is obtained from a multiplexer that selects among the carry to bit position 7 from the odd multiple generators (×3,×5, and ×7), the carry to bit position 6 from the multiple generator ×3 (to get the carry to position 7 of multiple×6), or carry zero for the other multiples. The resultant carry out is the selection signal used in the multiplexer of part A.

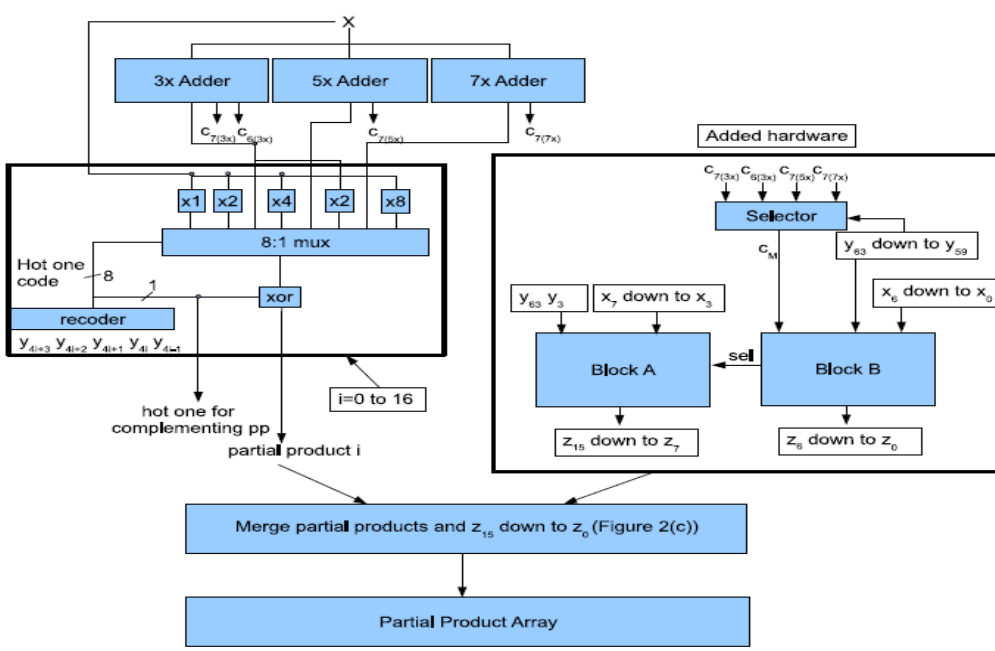



**Fig. 6. High level view of the recoding and partial product generation stage including our proposed scheme.**

TABLE I
TRUTH TABLE FOR COMPUTING THE CARRY OUT (– STANDS FOR "DON'T CARE")

| $C_M$ | $C_{out1}$ | $C_{out2}$ | $C_{out}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | - |
| 1 | 0 | 0 | - |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

For negative odd multiples we use a similar scheme. In this case the output of adder is complemented, but the only information available about the carry to position 7 is obtained directly from the adders that generate the positive odd multiple. Next, we show how to obtain the carry to the most significant part of the resultant complemented odd multiple from the carry to position 7 obtained from the adders. Let us call M the result of the positive odd multiple (output of the adder), and express M as

$$M = N + P \quad (1)$$

With P being the seven least-significant bits of the result from the adder, and N the remaining most significant bits of the result of the adder. Let us express N in terms of $C_7$ (carry to position 7)

$$N = Q + C_7 2^7 \quad (2)$$

That is, Q are the remaining most significant bits of the positive odd multiple minus the carry to position 7. Assuming a m bit partial product, the complement of M is expressed as

$$\overline{M} = 2^n - 1 - M = 2^n - 1 - N - C_7 2^7 - Q. \quad (3)$$

By adding and subtracting $2^7$ and rearranging terms results in

$$\overline{M} = 2^n - 2^7 - N - C_7 2^7 + 2^7 - 1 - Q. \quad (4)$$

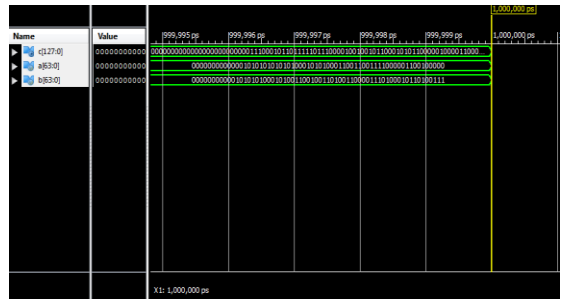
We identify the terms $N = 2^n - 2^7 - N$ and $Q = 2^7 - 1 - Q$. Taking into account these terms and adding and subtracting 27 and 2n−1 results in

$$\overline{M} = -2^{n-1} + \overline{N} + (2^{n-1} - 2^7) + (1 - C_7)2^7 + \overline{Q}. \quad (5)$$

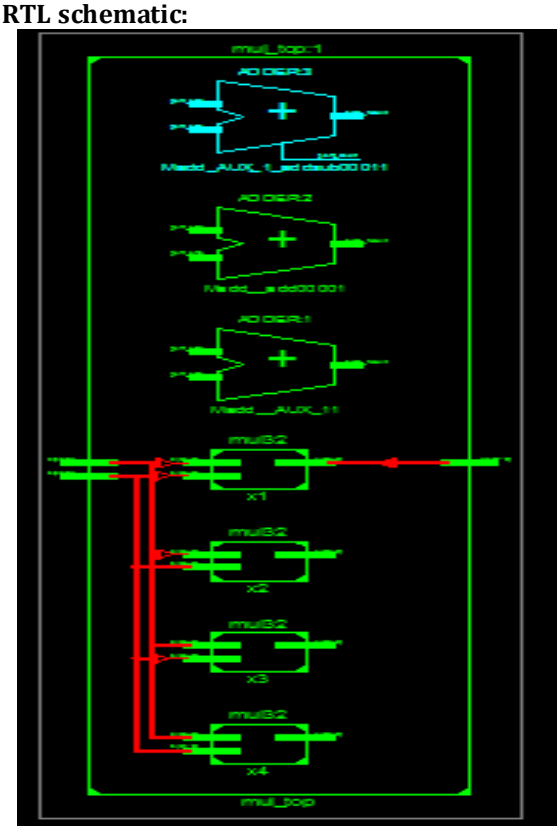
The term $(1 - C_7)2^7 + Q = C_7 + Q$ is computed in part B of the proposed scheme (see Fig. 5), but $(1 - C_7)2^7 = C_7$ is also part of the most significant part of partial product 15. Therefore, for a negative partial product we need to subtract $C_7$. In summary, we take CM as the carry to position 7 of the adder that generates the multiple when the partial product is positive, and complement this carry, when the partial product is negative.

## IV.RESULTS

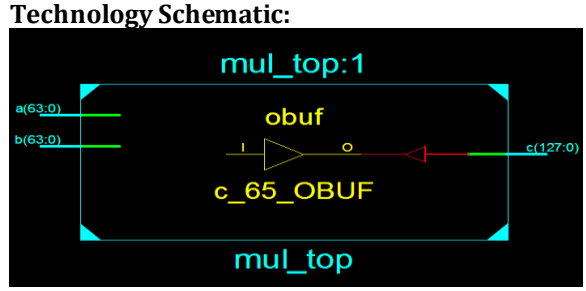
**Simulation results:**



**RTL schematic:**




**Technology Schematic:**




**Design summary:**

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| Logic Utilization | Used | Available | Utilization |
| Number of Slices | 2461 | 4656 | 52% |
| Number of 4 input LUTs | 4487 | 9312 | 48% |
| Number of bonded IOBs | 256 | 190 | 134% |
| Number of MULT18X18SIOs | 20 | 20 | 100% |

**Timing Report:**

```
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<23> (Madd__add0000_cy<23>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<24> (Madd__add0000_cy<24>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<25> (Madd__add0000_cy<25>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<26> (Madd__add0000_cy<26>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<27> (Madd__add0000_cy<27>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<28> (Madd__add0000_cy<28>)
   MUXCY:CI->O           1   0.051   0.000  Madd__add0000_cy<29> (Madd__add0000_cy<29>)
   MUXCY:CI->O           0   0.051   0.000  Madd__add0000_cy<30> (Madd__add0000_cy<30>)
   XORCY:CI->O           1   0.699   0.357  Madd__add0000_xor<31> (c_127_OBUF)
   OBUF:I->O                 3.169          c_127_OBUF (c<127>)
    ----------------------------------------
   Total                    37.236ns (29.856ns logic, 7.380ns route)
                                     (80.2% logic, 19.8% route)

==========================================================================
```

## V. CONCLUSION

In this paper, 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 ≥ 32 for a cell-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 overheads due to wiring.

## 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 T. Lang, Digital Arithmetic. Burlington, MA, USA: Morgan Kaufmann, 2004.
8. S. Vassiliadis, E. Schwarz, and D. Hanrahan, "A general proof for overlapped multiple-bit scanning multiplications," IEEE Trans. Comput., vol. 38, no. 2, pp. 172–183, Feb. 1989.
9. "Binary Multibit Multiplier," Patent 4 745 570 A, 1986.
10. D. Dobberpuhl et al., "A 200-MHz 64-b dual-issue CMOS microprocessor," IEEE J. Solid-State Circuits, vol. 27, no. 11, pp. 1555–1567, Nov. 1992.
11. E. M. Schwarz, R. M. A. III, and L. J. Sigal, "A radix-8 CMOS S/390 multiplier," in Proc. 13th IEEE Symp. Comput. Arithmetic (ARITH), Jul. 1997, pp. 2–9.
12. J. Clouser et al., "A 600-MHz superscalar floating-point processor," IEEE J. Solid-State Circuits, vol. 34, no. 7, pp. 1026–1029, Jul. 1999.
13. S. Oberman, "Floating point division and square root algorithms and implementation in the AMD-K7 microprocessor," in Proc. 14th IEEE Symp. Comput. Arithmetic (ARITH), Apr. 1999, pp. 106–115.
14. R. Senthinathan et al., "A 650-MHz, IA-32 microprocessor with enhanced data streaming for graphics and video," IEEE J. Solid-State Circuits, vol. 34, no. 11, pp. 1454–1465, Nov. 1999.
15. K. Muhammad et al., "Speed, power, area, latency tradeoffs in adaptive FIR filtering for PRML read channels," IEEE Trans. Very Large Scale Intgr. Syst., vol. 9, no. 1, pp. 42–51, Feb. 2001.
16. G. Colon-Bonet and P. Winterrowd, "Multiplier evolution: A family of multiplier VLSI implementations," Comput. J., vol. 51, no. 5, pp. 585–594, 2008.
17. R. Riedlinger et al., "A 32 nm, 3.1 billion transistor, 12 wide issue itanium processor for mission-critical servers," IEEE J. Solid-State Circuits, vol. 47, no. 1, pp. 177–193, Jan. 2012.
18. B. Cherkauer and E. Friedman, "A hybrid radix-4/radix-8 low power signed multiplier architecture," IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 44, no. 8, pp. 656–659, Aug. 1997.
19. D. Lutz, "ARM FPUs: Low latency is low energy," presented at the 22nd IEEE Symposium in Computer Arithmetic, Jun. 2015, [last visited Jul. 1, 2016]. [Online]. Available: http://arith22.gforge.inria.fr/slides/s1-lutz.pdf
20. V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach," IEEE Trans. Comput., vol. 45, no. 3, pp. 294–306, Mar. 1996.
21. Synopsys Inc., "Design Compiler," [Online]. Available: http://www. synopsys.com
22. "AX+2XAdder With Multi-Bit Generate/Propagate Circuit," Patent 5 875 125, 1997.