POLITECNICO DI TORINO
Repository ISTITUZIONALE

Improved 64-bit Radix-16 Booth Multiplier Based on Partial Product Array Height Reduction

Original
Improved 64-bit Radix-16 Booth Multiplier Based on Partial Product Array Height Reduction / Antelo, Elisardo; Montuschi, Paolo; Nannarelli, Alberto. - In: IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. I, REGULAR PAPERS. - ISSN 1549-8328. - ELETTRONICO. - 64:2(2017), pp. 409-418. [10.1109/TCSI.2016.2561518]

Availability:
This version is available at: 11583/2645860 since: 2021-04-06T22:32:55Z

Publisher:
IEEE Circuits and Systems Society

Published
DOI:10.1109/TCSI.2016.2561518

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

12 October 2024

Improved 64-bit Radix-16 Booth Multiplier Based on Partial Product Array Height Reduction

Elisardo Antelo, Paolo Montuschi, *Fellow, IEEE,* and Alberto Nannarelli, *Senior Member, IEEE*

***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 $\lceil n/4 \rceil$ for $n =$ 64-bit unsigned operands. This is in contrast to the conventional maximum height of $\lceil (n+1)/4 \rceil$. 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.**

***Index Terms*—Binary multipliers, radix-16, modified Booth recoding.**

I. Introduction

Binary multipliers are a widely used building block element in the design of microprocessors and embedded systems, and therefore, they are an important target for implementation optimization [1]–[6]. 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 non-redundant radix-r digits (by just making groups of m bits), and these are recoded from the set $\{0, 1, \ldots, r-1\}$ to the set $\{-r/2, \ldots, -1, 0, 1, \ldots, r/2\}$ to reduce the complexity of digit multiplications. For n-bit operands, a total of $\lceil n/m \rceil$ partial products are generated for two's complement representation, and $\lceil (n+1)/m \rceil$ for unsigned representation.

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 can not 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 $\{-8, -7, \ldots, 0, \ldots, 7, 8\}$, so that some odd multiples of the multiplicand have to be generated. Specifically, it is required to generate x3, x5 and x7 multiples (x6 is obtained by simple shift of x3). 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 are 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 a 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) [18] [4].

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] [2] some optimizations for radix-4 two's complement multipliers were introduced. Although for n-bit operands, a total of $\lceil n/2 \rceil$ partial products are generated, the resulting maximum height of the partial product array is $\lceil n/2 \rceil + 1$ 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

E. Antelo is with the Department of Electrónica e Computación, University of Santiago de Compostela, Santiago de Compostela, Spain.

P. Montuschi is with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy.

A. Nannarelli is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, Denmark.

Copyright ©2016 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org.

determine the delay and complexity of the reduction tree [7] [16]. In [1] [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 $\lceil n/2 \rceil$.

This reduction of one bit in the maximum height might be of interest for high-performance short-bit width 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: i) 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]; ii) 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.

The methods proposed in [1] [2] were mostly focused on two's complement radix-4 Booth multipliers, thus leaving open the research and extension to higher radices and unsigned multiplications (for unsigned integer arithmetic or mantissa times mantissa in a floating-point unit). For a radix higher than 4, it is necessary to generate the odd multiples (usually with adders), resulting in the reduction of the time slacks necessary to "hide" the simplified three bit assimilation. 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] [2].

In this work we present a technique that allows partial product arrays of maximum height of $\lceil n/m \rceil$ (with the goal of not increasing the delay of the partial product generation stage), for $r > 4$ and unsigned multipliers. Since for the standard unsigned multiplier the maximum height is $\lceil (n+1)/m \rceil$, 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 can not claim a successful result for all practical values of r and n and different implementation technologies. Thus, we concentrate on an specific instance: a 64-bit radix-16 Booth recoded unsigned multiplier implemented with a synthesis tool and a standard-cell library. We use 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).

The paper is structured as follows. Section II reviews the basic radix-16 Booth recoded multiplier. Section III describes the proposed method. Section IV presents the evaluation of our scheme, and Section V is devoted to the final conclusions.

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 x_i ($i = 0$ to $n-1$, with the least-significant bit, LSB, at position 0) and with Y the multiplier operand and bit components y_i.

The first step is the recoding of the multiplier operand [8]: groups of four bits with relative values in the set $\{0, 1, \ldots, 14, 15\}$ are recoded to digits in the set $\{-8, -7, \ldots, 0, \ldots, 7, 8\}$ (minimally redundant radix-16 digit set to reduce the number of multiples).

This recoding is done with the help of a transfer digit t_i and an interim digit w_i [7]. The recoded digit z_i is the sum of the interim and transfer digits:

$$z_i = w_i + t_i.$$

When the value of the four bits, v_i, is less than 8, the transfer digit is zero and the interim digit $w_i = v_i$. For values of v_i greater than or equal to 8, v_i is transformed into $v_i = 16 - (16 - v_i)$, so that a transfer digit is generated to the next radix-16 digit position (t_{i+1}) and an interim digit of value $w_i = -(16 - v)$ is left. That is:

$$\begin{array}{rllll} 0 \leq v_i < 8: & t_{i+1} = 0 & w_i = v_i & & w_i \in [0, 7] \\ 8 \leq v_i \leq 15: & t_{i+1} = 1 & w_i = -(16 - v_i) & & w_i \in [-8, -1] \end{array}$$

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 $\{-8, -7, \ldots, 0, \ldots, 7, 8\}$. Due to a possible transfer digit from the most significant radix-16 digit, the number of resultant radix-16 recoded digits is $\lceil (n+1)/4 \rceil$. 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 $\{-8, -7, \ldots, 0, \ldots, 7, 8\}$, 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

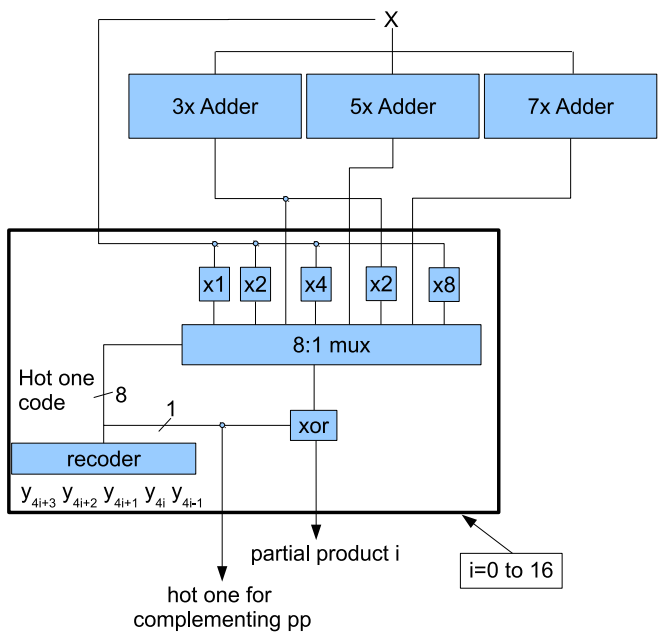

Fig. 1. Partial product generation.

multiples are obtained as before). Finally, 6X is obtained by a simple one bit left shift of 3X.

Figure 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).

Figure 2(a) illustrates part of the resultant bit array for $n = 64$ after the simplification of the sign extension [7].

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 Figure 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 (multioperand addition) from a maximum height of 17 (for $n = 64$) to 2 is performed. The methods for multioperand 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. Figure 2(b) shows the elements of the bit array to be added by the short adder. Figure 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 $n = 64$.

Figure 3 shows the specific elements of the bit array (boxes) to be added by the short carry-propagate addition. In this figure, $p_{i,j}$ corresponds to the bit j of partial product i, s_0 is the sign bit of partial product 0, $c_0 = NOT(s_0)$, b_i is the bit for the two's complement of partial product i, and z_i is the i-th 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 Figure 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 y_3 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 y_{63} (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]. Figure 4 shows the implementation of part A. The compound adder determines

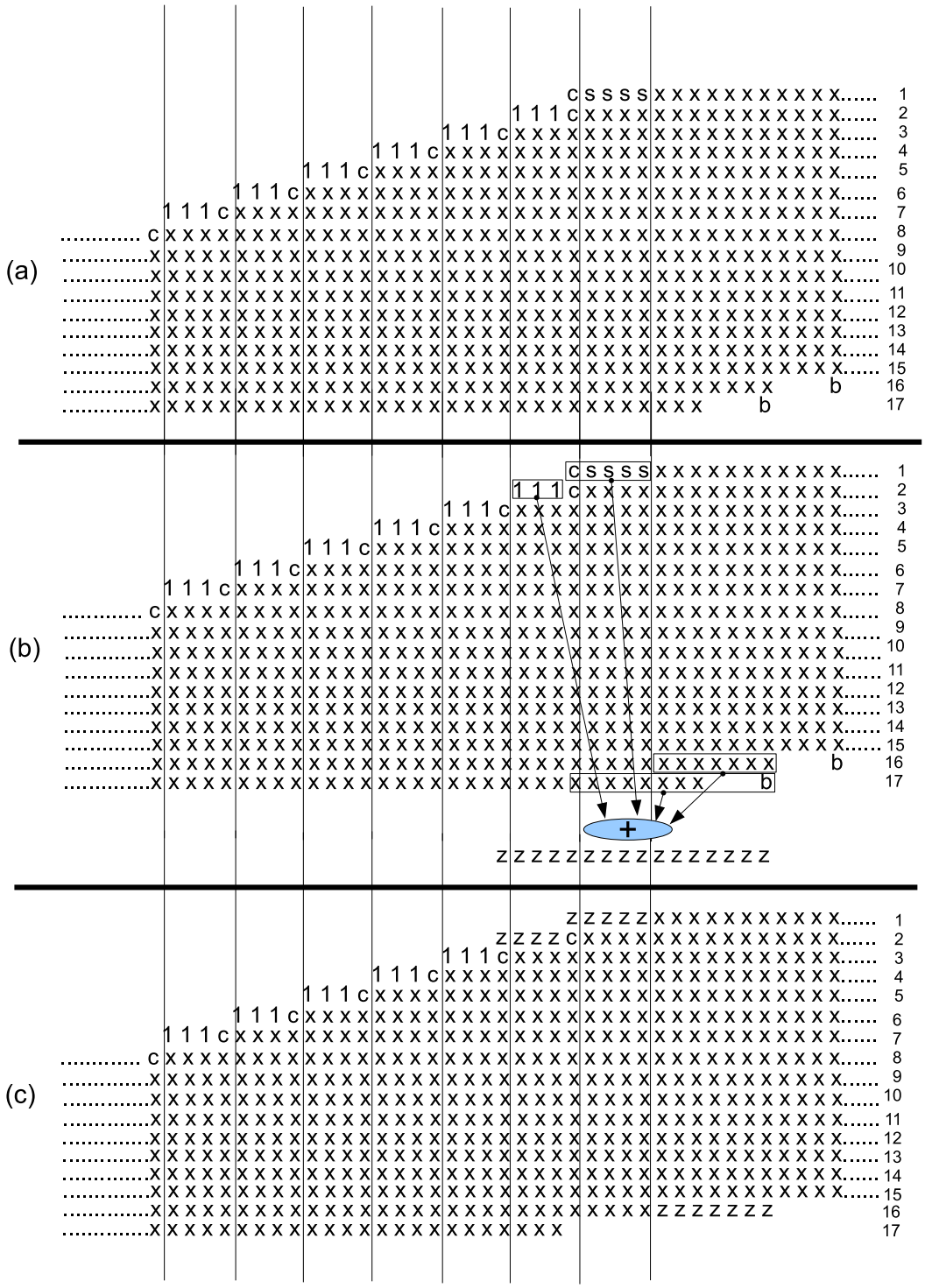

Fig. 2. Radix-16 partial product reduction array.

A									B							
	1	1	1	c_0	s_0	s_0	s_0	s_0	$p_{15,6}$	$p_{15,5}$	$p_{15,4}$	$p_{15,3}$	$p_{15,2}$	$p_{15,1}$	$p_{15,0}$
+				$p_{16,7}$	$p_{16,6}$	$p_{16,5}$	$p_{16,4}$	$p_{16,3}$	$p_{16,2}$	$p_{16,1}$	$p_{16,0}$				b_{15}
z_{15}	z_{14}	z_{13}	z_{12}	z_{11}	z_{10}	z_9	z_8	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0

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

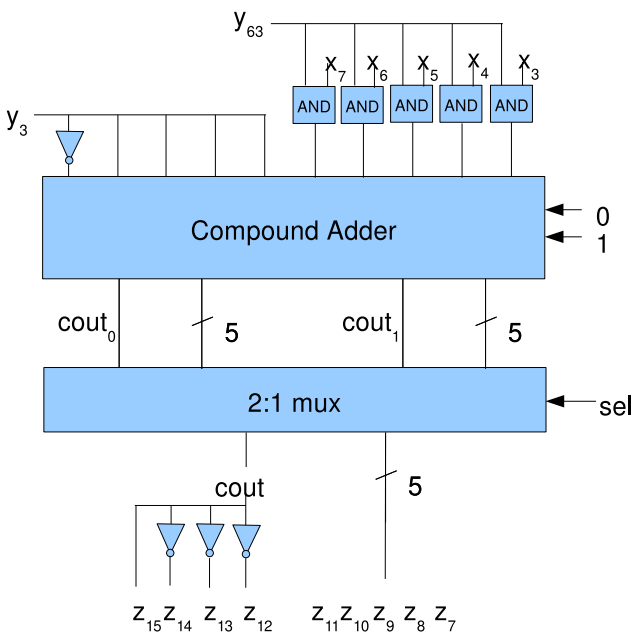

Fig. 4. Speculative addition of part A.

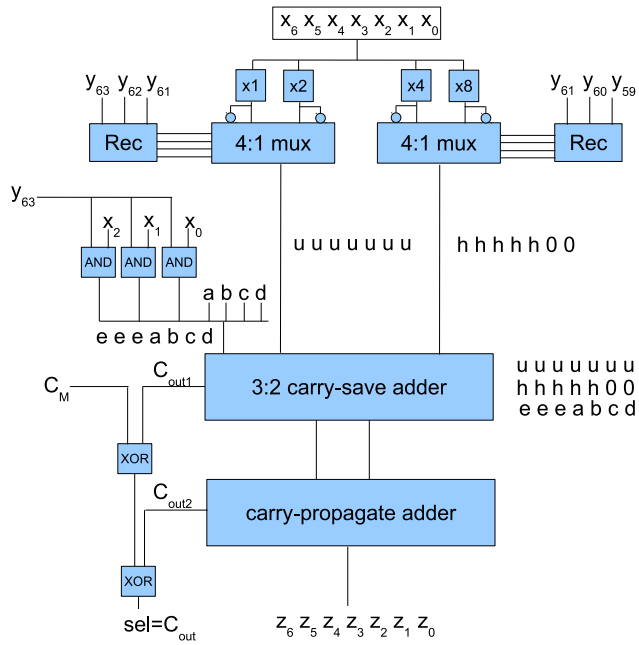

Fig. 5. Computation of part B.

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 b_{15}). 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.

Figure 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. 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 Figure 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.

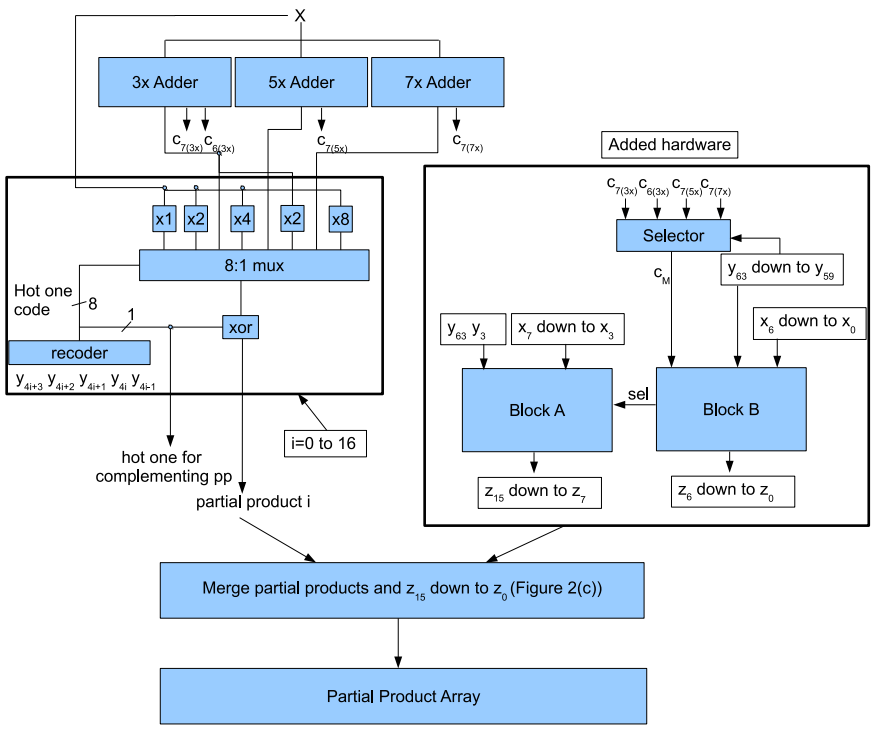

Fig. 6. High level view of the recoding and partial product generation stage including our proposed scheme.

Figure 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. Figure 5 shows that the computation of part B may generate two carry outs: the first from the 3:2 carry-save adder (C_{out1}), and the second from the carry-propagate adder (C_{out2}). To avoid inconsistencies, we detect the carry propagated to the most significant part of the partial product 15 (we call this C_M) and subtract it from the two carries generated in part B.

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

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 C_M carry is obtained from a multiplexer that selects among the carry to bit position 7 from the odd multiple generators (x3, x5 and x7), the carry to bit position 6 from the multiple generator x3 (to get the carry to position 7 of multiple x6), or carry zero for the other multiples. The resultant carry out is the selection signal used in the multiplexer of part A.

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

$$M = N + P \tag{1}$$

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):

$$N = Q + C_7 2^7 \tag{2}$$

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

$$\overline{M} = 2^n - 1 - M = 2^n - 1 - N - C_7 2^7 - Q \tag{3}$$

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 $\overline{N} = 2^n - 2^7 - N$ and $\overline{Q} = 2^7 - 1 - Q$. Taking into account these terms and adding and subtracting 2^7 and 2^{n-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 $(1 - C_7)2^7 + \overline{Q} = \overline{C_7} + \overline{Q}$ is computed in part B of the proposed scheme (see Figure 5), but $(1 - C_7)2^7 = \overline{C_7}$ is also part of the most significant part of partial product 15. Therefore, for a negative partial product we need to subtract $\overline{C_7}$.

In summary, we take C_M 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. Evaluation

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.

A. 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 μm^2. We synthesized the full partial product generation stage for the basic scheme allowing Synopsys' DesignWare [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 DesignWare in this case). We did not optimize the 3X adder as described for instance in [12], [22], [23], since this optimization can not be applied to the 5X and 7X adders, so that the critical path remains the same.

Figure 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 2K 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 1.5K NAND-2 gates corresponds to the penalty of not using DesignWare adders in the proposed design.

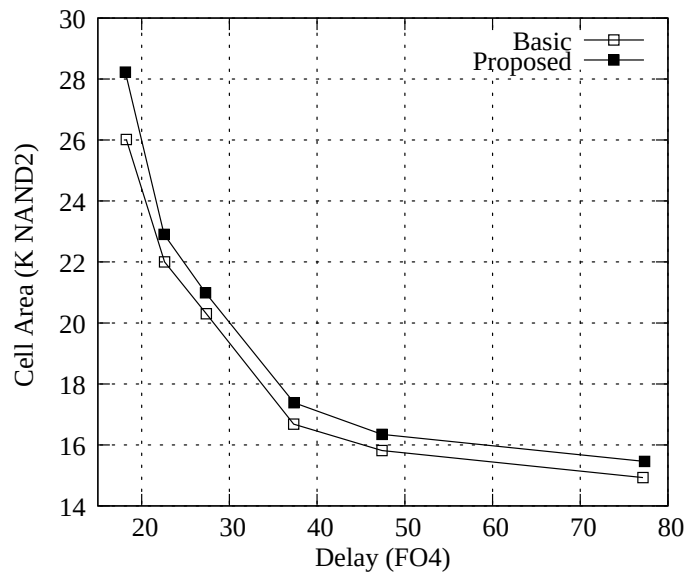

Fig. 7. Latency-area space for the partial product generation stage: basic scheme vs proposed scheme.

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 (Fig. 8.a) with pipeline register placed between the partial products generation (stage

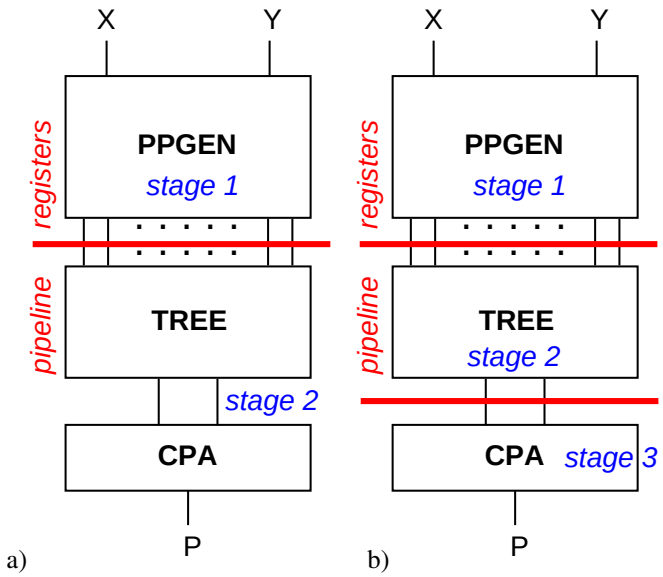

Fig. 8. Pipelined multiplier. a) 2-stage; b) 3-stage.

abbreviated as PPGEN in the figures and tables) and the tree (TREE);

2) a 3-stage design (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. This may result in increased number of flip-flops (e.g., inside the tree) and it is also non-suitable for reuse. Standard datapath blocks (e.g., CPAs) are normally taken from fully-tested hardware libraries and altering their behavior (placing pipeline registers inside) will prolong development times, re-validation and re-testing.

1) Design of 2-Stage Multiplier: For the 2-stage multiplier the critical path lies in the second stage for both the *basic* and the *proposed* multipliers. The delay of the critical path is 23 FO4 for the *basic* and 21.5 FO4 for the *proposed* multiplier.

Clearly, the reduced number of partial products in the *proposed* unit at the tree input (16 vs. 17 operands) makes the accumulation faster. The area of the 2-stage implementation it is slightly larger for the *proposed* multiplier, as shown in Fig. 9.

As for the power dissipation, Table II reports the power breakdown for the main blocks of the pipelined multiplier. The proposed unit consumes about 2% less power than the basic unit. This is mostly due to the reduced switching activity (glitches) in the second stage (tree and CPA).

2) Design of 3-Stage Multiplier: The maximum throughput for the multiplier can be obtained by breaking the critical part of the second stage in two stages. To minimize the number of flip-flops, or latches, this second register is placed between the tree and the CPA.

With this pipelining, the critical path lies in the first stage of the multiplier for both the *basic* and the *proposed* multipliers.

As already shown in Sec. IV-A, the delay of the critical path is 18 FO4 for both implementations.

Fig. 9. Area breakdown for 2-stage pipelined multipliers.

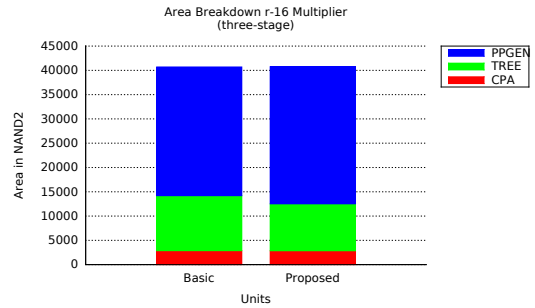

Fig. 10. Area breakdown for 3-stage pipelined multipliers.

TABLE II
POWER DISSIPATION IN THE PIPELINED MULTIPLIERS.

	TWO-STAGE				THREE-STAGE			
	BASIC		PROPOSED		BASIC		PROPOSED	
	[mW]	%	[mW]	%	[mW]	%	[mW]	%
PPGEN	1.93	28	1.95	29	3.29	46	2.97	43
TREE	2.44	35	2.29	34	1.29	18	1.27	19
CPA	0.51	7	0.47	7	0.23	3	0.23	3
REGs	1.99	29	2.02	30	2.34	33	2.37	35
TOTAL	6.86	100	6.72	100	7.15	100	6.84	100

Power is measured at 100 MHz frequency.

In this case, the larger slack[1] in stage 2, allows for a good reduction in area for the tree of the *proposed* multiplier, that partly compensate the larger area in the first stage (Fig. 7). As a result, the area of the two units in the 3-stage implementation is almost the same, as reported in Fig. 10.

Also in this case, the power dissipation is slightly (4%) lower in the proposed unit. The breakdown of the different parts is reported in Table II.

C. High Level Evaluation

In this subsection we use a high level rough model to evaluate the proposed method. We evaluate the critical path of the conventional partial product generation and the critical path of the hardware we added to reduce the maximum height of the partial product array. Although real implementations rely on optimizations of the critical path done by synthesis tools

[1]The synthesizer trades-off slack for smaller area and lower power dissipation gates. Therefore, the actual slack reported by the static timing analysis is close to zero for the three stages.

TABLE III
DELAY EQUATIONS, INPUT CAPACITANCE AND HARDWARE COST OF BASIC ELEMENTS.

Element	Delay # FO4	Input Capacitance # inverters	Area # NAND2
NAND2	$0.4 + 0.2L$	$4/3$	1.0
NOR2	$0.4 + 0.2L$	$5/3$	1.3
INV	$0.2 + 0.2L$	1	0.4
AOI12	$0.47 + 0.2L$	$(5/3, 2, 2)$	2.1
OAI12	$0.53 + 0.2L$	$(4/3, 2, 2)$	2.0
XOR	$0.9 + 0.2L$	$7/3$	2.5
MUX2	$0.9 + 0.2L$	$(data : 4/3, sel : 7/3)$	2.5
MUX4	$1.5 + 0.2L$	$(data : 4/3, sel : 7/3)$	5.0
Full-Adder	$a, b : 2.73 + 0.2L$ $c : 0.9 + 0.2L$	$(a : 11/3, b : 7/3)$ $11/3$	7.5
Buffer	$0.72 \ln(L/L_{in})$	L_{in}	$\ln(L/L_{in})$

on a specific standard cell library technology, this high level analysis may give some insight about the relative contribution to the critical path of each component.

We use a rough delay model based on logical effort [24]. This model is based on using cells with transistor sizing so that all the cells have the drive strength of the minimum size inverter. Buffering is introduced when necessary to optimize delays. We provide delays in FO4 units. Interconnections loads are not taken into account. Optimizations such as gate sizing, low/high V_{th}, etc. are not considered.

Table III shows the delay equations, input capacitance and relative hardware cost of the basic hardware elements used. In the table, the parameter L indicates the actual load (capacitance) connected to the specific gate, and L_{in} indicates the input capacitance of the buffers.

A key issue for the estimation of the critical path of the conventional partial product generation is the architecture of the adders for multiple generation. The worst case for our analysis corresponds to the fastest design point for partial product generation. Therefore we considered a fast Kogge-Stone adder topology [7]. Although this is not energy/power efficient in real implementations, at the logic level it is a good lower bound of delay for an adder.

After the analysis of the conventional architecture, we estimated the impact of the additional hardware required for the proposed multiplier. For a quick reference, the timing paths of Figures 4, 5 and 6 are summarized in Fig. 11. In the figure the delay in the input registers (X and Y) and the delay of buffers are omitted for simplicity.

The critical path of the conventional partial product generation is composed by the following items:

- Input register X: 3.0 FO4.
- Input buffering of multiplicand: 1.4 FO4.
- Multiple generation (adder): 10.3 FO4.
- Buffer between multiple generators and 8:1 mux: 1.7 FO4.
- 8:1 mux and inversion (input from data): 4.9 FO4.

This corresponds to a total delay of 21.3 FO4 in the critical path.

The scheme we propose (Part A, Fig. 4, and Part B, Fig. 5, in Fig. 11) has the following components in the critical path:

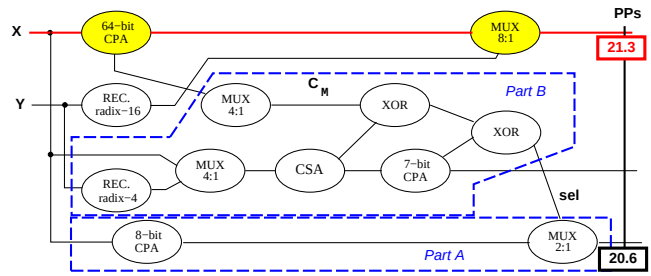

Fig. 11. Timing paths for the proposed partial products reduction.

- Input register Y: 3.0 FO4.
- Input buffering of multiplier bits: 0.5 FO4.
- Radix-4 Booth recoding and selector with inversion (Part B): 5.0 FO4.
- 3:2 carry-save adder (Part B): 3.5 FO4.
- Carry out of 7-bit carry-propagate adder (Part B): 4.4 FO4.
- XOR to produce *sel* signal (Part B) and six-bit 2:1 multiplexer (Part A): 4.2 FO4.

Thus, the path delay is 20.6 FO4 and it is not critical.

Our analysis shows that the C_M signal is not in the critical path (the worst case delay for C_M is 13.1 FO4, while the worst case delay for C_{out2} is 16.4 FO4).

These results are coherent with the fastest design point in the latency-area graph shown in Figure 7. The lower bound in latency is about 18.2 FO4. The synthesis tool is able to do some sort of gate sizing (dependent on the available gate sizes for each instance gate), so a faster result than in our high level analysis should be expected.

Therefore, our rough analysis is in agreement with the synthesis results, as the proposed scheme is not in the critical path for $n = 64$.

We performed a similar analysis for the critical path of the conventional partial product generation for $n = 32$ (the case for $n = 16$ is less attractive for radix-16 due to the small number of partial products). For $n = 32$ we obtain a critical path of 19.7 FO4. As it can be seen in Figure 2, the scheme we propose is not sensitive to the variation of n (the number of bits added in Figure 2(b) is independent of the value of n), thus resulting in the same critical path as before (20.6 FO4). Therefore, for the fastest design point, for $n = 32$, the proposed scheme is in the critical path, with a slack with respect to the conventional partial product generation of 0.9 FO4 for $n = 32$. This negative slack of our scheme can be reduced with conventional approaches like low V_{th} gates and gate sizing without significant increase in power, since the share of our scheme with respect to the total hardware is very small.

We verified this statement with the synthesis tool. A synthesis for $n = 32$ leads to a critical path of 16.5 FO4 and this critical path corresponds to the computation of a regular partial product.

V. Conclusions

Pipelined large wordlength 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 $n \geq 32$ 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 Transactions on Circuits and Systems II: Express 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 Transactions on Computers*, vol. 60, no. 2, pp. 148–156, Feb. 2011.

[3] N. Petra *et al.*, "Design of Fixed-Width Multipliers With Linear Compensation Function," *IEEE Transactions on Circuits and Systems I: Regular 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 Symposium on Computer 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 Transactions on Circuits and Systems I: Regular 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 Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 12, pp. 3426–3435, Dec. 2014.

[7] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[8] S. Vassiliadis, E. Schwarz, and D. Hanrahan, "A General Proof for Overlapped Multiple-Bit Scanning Multiplications," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 172–183, Feb. 1989.

[9] "Binary multibit multiplier," U.S. Patent 4 745 570 A, 1986.

[10] D. Dobberpuhl *et al.*, "A 200-MHz 64-b Dual-Issue CMOS Microprocessor," *IEEE Journal of 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 Symposium on Computer Arithmetic (ARITH)*, July 1997, pp. 2–9.

[12] J. Clouser *et al.*, "A 600-MHz Superscalar Floating-Point Processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 7, pp. 1026–1029, July 1999.

[13] S. Oberman, "Floating point division and square root algorithms and implementation in the AMD-K7 microprocessor," in *Proc. 14th IEEE Symposium on Computer 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 Journal of Solid-State Circuits*, vol. 34, no. 11, pp. 1454–1465, Nov. 1999.

[15] K. Muhammad *et al.*, "Speed, Power, Area, and Latency Tradeoffs in Adaptive FIR Filtering for PRML Read Channels," *IEEE Transactions on VLSI Systems*, vol. 9, no. 1, pp. 42–51, Feb. 2001.

[16] G. Colon-Bonet and P. Winterrowd, "Multiplier Evolution: A Family of Multiplier VLSI Implementations," *The Computer Journal*, 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 Journal of 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 Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 8, pp. 656–659, Aug. 1997.

[19] D. Lutz and N. Burgess, "Low Latency is Low Energy," in *Proc. 48th Asilomar Conf. on Signals, Systems and Computers*, Nov. 2014.

[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 Transactions on Computers*, vol. 45, no. 3, pp. 294–306, Mar. 1996.

[21] Synopsys Inc. Design Compiler. [Online]. Available: http://www.synopsys.com

[22] "A X+2X adder with multi-bit generate/propagate circuit," U.S. Patent 5 875 125, 1997.

[23] "3x adder," U.S. Patent 6 269 386 B1, 1998.

[24] A. Vazquez and E. Antelo. (2012, June) Area and Delay Evaluation Model for CMOS Circuits. Internal Report, Univ. of Santiago de Compostela. [Online]. Available: http://www.ac.usc.es/node/1607