Implementation of Radix-4 in 2's Complement Modified Booth Encoded Multiplier

Article · September 2013

1 author:

David Solomon Raju
HITS COE
12 PUBLICATIONS **3** CITATIONS

SEE PROFILE

K.RaghuRam.et, al., International Journal of Technology and Engineering Science [IJTES] ™
Vol 1 (6), pp 609 – 617, September 2013

Implementation of Radix-4 in 2's Complement Modified Booth Encoded Multiplier

K. Raghuram [1], B. Shanthi [2], David Solomon Raju Y[3]

[1] *Assoc .Prof.ECE, Pragathi Engineering College, Kakinada, East Godavari Dt.-533 437 A.P.*
[2] *PG Student, M. Tech Pragathi Engineering College, Kakinada, East Godavari Dt.-533 437 A.P.*
[3]*Assoc. Prof. ECE, Holy Mary Institute of Technology & Science, Keesara, R. R. Dt.- 501 301, A.P.*

Abstract *- In this paper, we present a technique to reduce by one row the maximum height of the partial product array generated by a radix-4 Modified Booth Encoded multiplier, without any increase in the delay of the partial product generation stage. This reduction may allow for a faster compression of the partial product array and regular layouts. This technique is of particular interest in all multiply designs, but especially in short bit-width two's complement multipliers for high-performance embedded cores. Twos complement multipliers are important for a wide range of applications. The proposed method is general and can be extended to higher radix encodings, as well as is used for higher radices encoding for any size of m × n multiplications this reduction may allow for a faster compression of the partial product array and regular layouts. This technique is of particular interest in all multiplier designs, but especially in short bit-width two's complement multipliers for high-performance embedded cores. With the extra hardware of a (short) 3-bit addition, and the simpler generation of the first partial product row can be achieved. Implementation is done by using Xilinx for synthesis and modelsim for simulation in HDL.*

Keywords *- Multiplication, Modified Booth encoding, partial product array, Radix-4.*

I. INTRODUCTION

In multimedia, 3D graphics and signal processing applications, performance, in most cases, strongly depends on the effectiveness of the hardware used for computing multiplications, since multiplication is, besides addition, massively used in those environments. The high interest in this application field is witnessed by the large amount of algorithms and implementations of the multiplication operation, which have been proposed in the literature (for a representative set of references, see [1]). More specifically, short bit-width (8-16 hits) two's complement multipliers With single-cycle throughput and latency have emerged and become very important building blocks for high-performance embedded processors and DSP execution cores [?.], [3j. !n this case, the multiplier must be highly optimized to fit within the required cycle lime and power budgets. Another relevant application for short bit width multipliers is the design of SIMD units supporting different data formats [3], |4]. In this case, short bit-width multipliers often play the role of basic building blocks.

Two's complement multipliers of moderate bit-width (less than 32 bits) are also being used massively in FT'GAs. All of the above translates into a high interest and motivation on the part of the industry, for the design of high-performance short or moderate bit-width two's complement multipliers.

The basic algorithm for multiplication is based on the well-known paper and pencil approach [1] and passes through three main phases: 1) partial product (PP) generation, 2) PP reduction, and 3) final (carry-propagated) addition. During PP generation, a set of rows is generated where each one is the result of the product of one bit of the multiplier by the multiplicand. For example, if we consider the multiplication x with both X and Y on n bits and of the form $x_{n-1} \ldots\ldots r_o$ then the 7th row is, in general, a proper left shifting of $y_i x X$, i.e., either a string of all zeros when $y_i = 0$, or the multiplicand A itself when. y, = I. In this case, the number of PP rows generated during the first phase is clearly n.

Modified Booth Encoding (MBE) [5] is a technique that has been introduced to reduce the number of PP rows, still keeping the generation process of each row both simple and fast enough. One of the most commonly used schemes is radix-4 MBE, for a number of reasons, the most important being that it allows for the reduction of the size of the partial product array by almost half, and it is very simple to generate the multiples of the multiplicand. More specifically, the classic two's complement n x n bit multiplier using the radix-4 MBE scheme, generates a PP array with a maximum height of + 1 rows, each row before the last one being one of the following possible values: all zeros, $\pm$x, $\pm$2X. The last row, which is due to the negative encoding, can be kept very simple by using specific- techniques integrating two's complement and sign extension prevention [1].

The PP reduction is the process of adding all PP rows by using a compression tree [61, [7]. Since the knowledge of 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.

TABLE – 1: Modified Booth Encoding (Radix-4)

y_{2i+1}	y_{2i}	y_{2i-1}	Generated partial products
0	0	0	$0 \times X$
0	0	1	$1 \times X$
0	1	0	$1 \times X$
0	1	1	$2 \times X$
1	0	0	$(-2) \times X$
1	0	1	$(-1) \times X$
1	1	0	$(-1) \times X$
1	1	1	$0 \times X$

In this work, we introduce an idea to overlap, to some extent, the PP generation and the PP reduction phases. Our aim is to produce a PP array with a maximum height of rows that is then reduced bv the compressor tree stage. As we will see for the common case of values n which are power of two, the above reduction can lead to an implementation where the delay of the compressor tree is reduced by one XOR2 gate keeping a regular layout. Since we are focusing on small values of n and fast single-cycle units, this reduction might be important in cases where, for example, a high computation performance through the assembly of a large number of small processing units with limited computation capabilities is required, such as 8x 8 or 16 x 16 multipliers [8]. A similar study aimed at the reduction of the maximum height to but using a different approach has recently presented interesting results in [9] and previously, by the same authors, in [10]. Thus, in the following, we will evaluate and compare the proposed approach with the technique in [9]. Additional details of our approach, besides the main results presented here, can be found in [11].

The paper is organized as follows: in Section 2, the multiplication algorithm based on MBE is briefly reviewed and analyzed. In Section 3, we describe related works. In Section 4, we present our scheme to reduce the maximum height of the partial product array by one unit during the generation of the PP rows. Finally, in Section 5, we provide the simulated and synthesis results followed by conclusion of the paper.

II. MODIFIED BOOTH RECODED MULTIPLIERS

In general, a radix-B = 2^b MBE leads to a reduction of the number of rows to about [n/b] while, on the other hand, it introduces the need to generate all the multiples of the multiplicand X, at least from *-B/2* x *X* to B/2 x *X*. As mentioned above, radix-4 MBE is particularly of interest since, for radix-4, it is easy to create the multiples of the multiplicand 0, ±*X*, ±*2X*. In particular, ±*2X* can be simply obtained by single left shifting of the corresponding terms ±*X*. It is clear that the MBE can be extended to higher radices (see [12] among others), but the advantage of getting a higher reduction in the number of rows is paid for by the need to generate more multiples of *X*. In this paper, we focus our attention on radix-4 MBE, although the proposed method can be easily extended to any radix-B MBE [11].

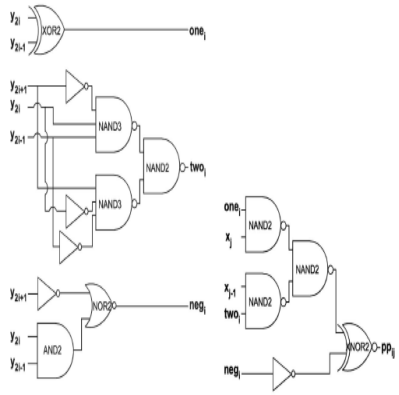


Fig. 1 Gate-level diagram for partial product generation using MBE (adapted from [10]). (a) MBE signals generation. (b) Partial product generation.

From an operational point of view, it is well known that the radix-4 MBE scheme consists of scanning the multiplier operand with a three-bit window and a stride of two bits (radix-4). For each group of three bits (y_{2i+1}, y_{2i}, y_{2i-1}) only one partial product row is generated according to the encoding in Table 1. A possible implementation of the radix-4 MBE and of the corresponding partial product generation is shown in Fig. 1, which comes from a small adaptation of [10, Fig. 12b). For each partial product row Fig. 1a, produces the *one, two,* and *neg* signals These signals are then exploited by the logic in Fig. lb, along with the appropriate bits of the multiplicand, in order to generate the whole partial product array. Other alternatives for the implementation of the recoding and partial product generation can be found in [13], [14], [15], among others.

As introduced previously, the use of radix-4 MBE allows for the (theoretical) reduction of the PP rows to, with the possibility for each row to host a multiple of y_i x x with y_i ε {*0*, ±1, ±2. While it is straight forward to generate the positive terms 0, *X*, and 2X at least through a left shift of X, some attention is required to generate the terms *X* and *-2X* which, as observed in Table 1, can arise from three configurations of the y_{2i+1}, $y_{2i,}$ and $y_{2i,-1}$, 1 bits. To avoid computing negative encodings, i.e., - *X* and *-2X*, the two's complement of the multiplicand is generally used. From a mathematical point of view, the use of two's complement requires extension of the sign to the leftmost part of each partial product row, with the

consequence of an extra area overhead. Thus, a number of strategies for preventing sign extension have been developed. The array resulting from the application of the sign extension prevention technique in [1] to the partial product array of a 8 x 8 MBE multiplier [5] is shown in Fig. 2. The use of two's complement requires a *neg* signal (e.g., neg_0, neg_1, neg_2 *and* neg_3, in Fig. 2) to be added in the LSB position of each partial product row for generating the two's complement, as needed. Thus, although for an *n x n multiplier*, only [n/2] partial products are generated, the maximum height of the partial product array is [n/2] +1.

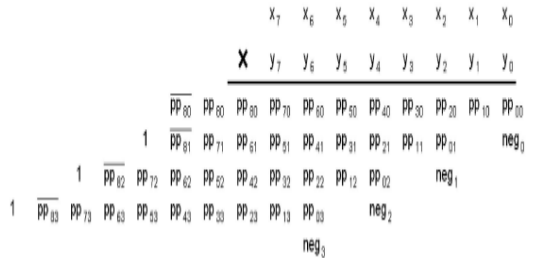


Fig. 2 Application of the sign extension prevention measure [1] on the partial product array of a 8 × 8 radix-4 MBE multiplier

When 4-to-2 compressors are used, which is a widely used option because of the high regularity of the resultant circuit layout for *n* power of two, the reduction of the extra row may require an additional delay of two XOR2 gates. By properly connecting partial product rows and using a Wallace reduction tree [7], the extra delay can be further reduced to one XOR2 [16], [17]. However, the reduction still requires additional hardware, roughly a row of *n* half adders. This issue is of special interest when *n* is a power of two, which is by far a very common case, and the multiplier's critical path has to fit within the clock period of a high performance processor. For instance, in the design presented in [2], for *a* 10, the maximum column height of the partial product array is nine, with an equivalent delay for the reduction of six XOR2 gates [16], [17]. For a maximum height of the partial product array of 8, the delay of the reduction tree would be reduced by one XOR2 gate [16], 117]. Alternatively, with a maximum height of eight, it would be possible to use 4 to 2 adders, with a delay of the reduction tree of six XOK2 gates, but with a very regular layout.

III. PROPOSED WORK

Some approaches have been proposed aiming to add the [n/2] +1 rows, possibly in the same time as the [n/2] rows. The solution presented in 14 is based on the use of different types of counters, that is, it operates at the level of the PP reduction phase. Kang and Gaudiot propose a different approach in [9] that manages to achieve the goal of eliminating the extra row before the PP reduction phase. This approach is based on computing the two's complement of the last partial product, thus eliminating the need for the last *neg* signal, in a logarithmic time complexity. A special tree structure (basically an incremented implemented as a prefix tree [18]) is used in order to produce the two's complement (Fig. 3), by decoding the MBE signals through a *3-5* decoder (Fig. 4a).

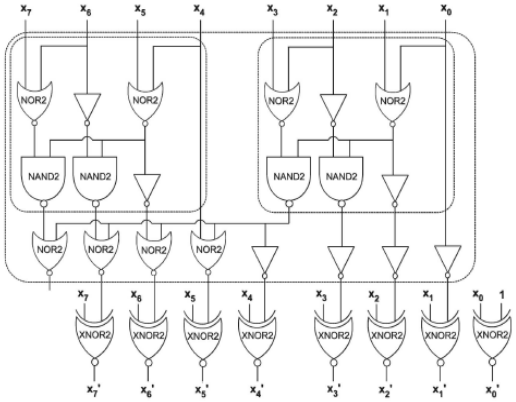


Fig. 3 Two's complement computation (n = 8) [9].

Finally, a row of 4-1 multiplexers with implicit zero output' is used (Fig. 4b) to produce the last partial product row directly in two's complement, without the need for the *neg* signal.

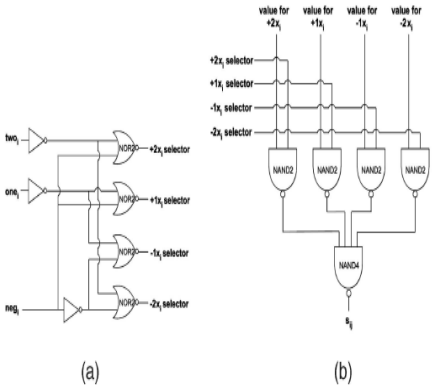


Fig. 4 Gate-level diagram for the generation of two's complement partial product rows [9]. (a) 3-5 decoder. (b) 4-1 multiplexer

The goal is to produce the two's complement in parallel with the computation of the partial products of the other rows with maximum overlap. In such a case, it is expected to have no or a small time penalization in the critical path. An example of the partial product array produced using the above method is depicted in Fig. 5.

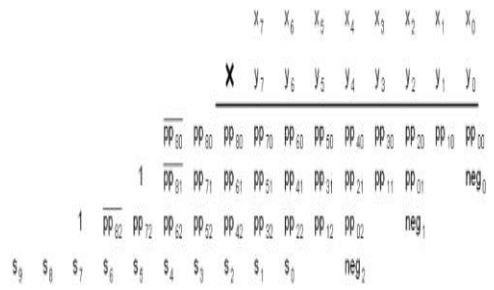


Fig. 5 Partial product array by applying the two's complement computation method in [9] to the last row.

In this work, we present a technique that also aims at producing only [n/2] rows, but by relying on a different approach than [9].

IV. DESIGN IDEA

The case of *n* x *n* square multipliers is quite common, as the case of *n* that is a power of two. Thus, we start by focusing our attention on square multipliers, and then present the extension to the general case, of *m x n* rectangular multipliers.

A. Square Multipliers

The proposed approach is general and, for the sake of clarity, will be explained through the practical case of 8×8 multiplication (as in the previous figures). As briefly outlined in the previous sections, the main goal of our approach is to produce a partial product array with a maximum height [n/2] of rows, without introducing any additional delay.

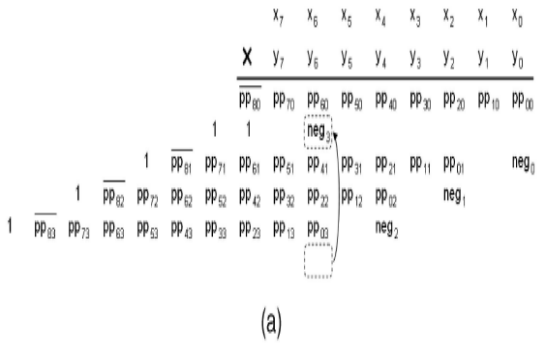


Fig. 6 Partial product array after adding the last neg bit to the first row. (a) Basic idea. (b) Resulting array

Let us consider, as the starting point, the form of the simplified array as reported in Fig *7.* for all the partial product rows except the first one As depicted in Fig. 6a, the first row is temporarily considered as being split into two sub rows, the first one containing the partial product bits (from right to left) from pp_{00}, to pp_{80}' and the second one with two bits set at "one" in positions 9 and 8 Then, the bit *in* neg_3 related lo the fourth partial product row, is moved to become a part of the second sub row. The key point of this "graphical" transformation is that the second sub row containing also the bit neg_3, can now be easily added to the first sub row, with a constant short carry propagation of three positions (further denoted as "3-bits addition"), a value which is easily shown to be general, i.e., independent of the length of the operands, for square multipliers. In fact, with reference to the notation of Fig. 6 we have that $qq_{90} \cdot qq_{90} qq_{80} qq_{70} qq_{60} = 0\,0\ pp_{80} \cdot pp_{70}\ pp_{60} + 0\ 1\ 1\ 0\ neg_3$. As introduced above, due to the particular value of the second operand, i.e., 0 1 1 0 neg_3, in [11], we have observed that it requires a carry propagation only across the least-significant three positions, a fact that can also be seen by the implementation shown in Fig. 7.

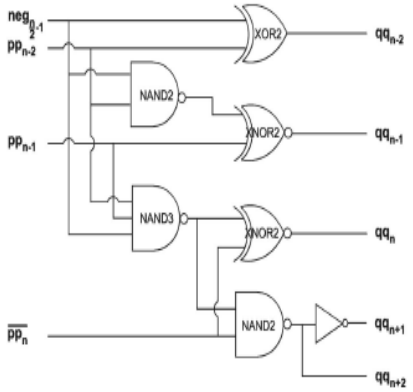


Fig. 7 Gate-level diagram of the proposed method for adding the last neg bit in the first row.

It is worth observing that, in order not to have delay penalizations, it is necessary that the generation of the other rows is done in parallel with the generation of the first row cascaded by the computation of the bits $qq_{90}\,qq_{80}\,qq_{70}\,qq_{60}$ in Fig. 6b. In order to achieve this, we must simplify and differentiate the generation of the first row with respect to the other rows. We observe that the Booth recoding for the first row is computed more easily than for the other rows, because the *y*. bit used by the MBE is always equal to zero. In order to have a preliminary analysis which is possibly independent of technological details, we refer to the circuits in the following figures:

- Fig. 1, slightly adapted from [10, Fig. 12], for the partial product generation using MBE;
- Fig. 7, obtained through manual synthesis (aimed at modularity and area reduction without compromising the delay), for the addition of the last *neg* bit to the three most significant bits of the first row;
- Fig. 8, obtained by simplifying Fig. 1 (since, in the first row, it is y_{2i-1} , =0), for the partial product generation of the first row only using MBE; and

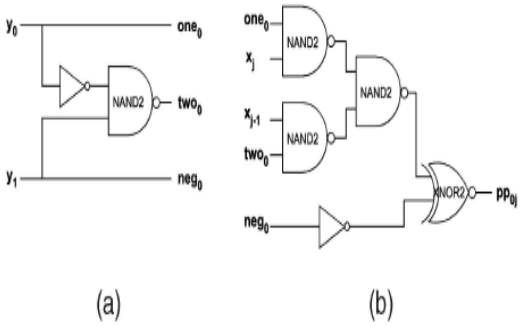


Fig. 8 Gate-level diagram for first row partial product generation. (a) MBE signals generation. (b) Partial product generation.

- Fig. 9, obtained through manual synthesis of a combination of the two parts of Fig. 8 and aimed at decreasing the delay of Fig. 8 with no or very small area increase, for the partial product genera-lion of the first row only using MBE.

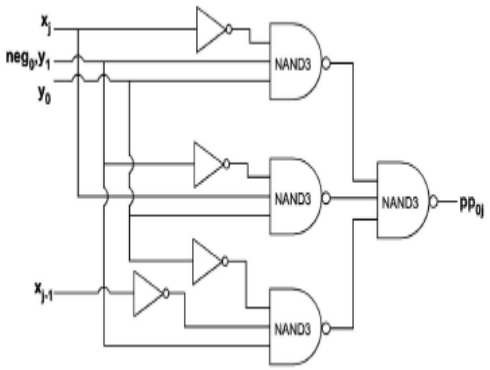


Fig. 9 Combined MBE signals and partial product generation for the first row (improved for speed).

In particular, we observe that, by direct comparison of Figs. 1 and 8, the generation of the MBE signals for the first row is simpler, and theoretically allows for the saving of the delay of one NAND3 gate. In addition, the implementation in Fig. 9 has a delay that is smaller than the two parts of Fig. 8, although it could require a small amount of additional area. As we see in the following, this issue hardly has any significant impact on the overall design, since this extra hardware is used only for the three most significant bits of the first row, and not for all the other bits of the array.

The high-level description of our idea is *as* follows:

1. generation of the three most significant bit weights of the first row, plus addition of the last *neg* bit: possible implementations can use a replication of three times the circuit of Fig. 9 (each for the three most significant bits of the first row), cascaded by the circuit of Fig. 7 to add the *neg* signal;
2. parallel generation of the other bits of the first row: possible implementations can use instances of the circuitry depicted in Fig. 8, for each bit of the first row, except for the three most significant;
3. parallel generation of the bits of the other rows: possible implementations can use the circuitry of Fig. 1, replicated for each bit of the other rows

All items 1 to 3 are independent, and therefore can be executed in parallel clearly if, as assumed and expected, item 1 is not the bottleneck (i.e., the critical path), then the implementation of the proposed idea has reached the goal of not introducing time penalties.

TABLE 2: Designs for the Generation of the Partial Product Rows Considered in the Evaluation

Implementation	Description	Motivation
Standard multiplier (any row)	Standard implementation of the MBE: signals *one, two, and neg* generated first and then used to produce the partial product array (Fig. 1)	The delay to generate a generic partial product row (other than the first one) in a standard *n x n* multiplier represents the upper bound for any design aimed at removing the last *neg* signal (either working on the first or last row.
Standard multiplier (first row)	Alternative implementation of the MBE for the first row: logic for the generation of he partial product row is simplified as the y_{-1} bit is always equal to zero (fig. 8)	The delay to produce the first row constitutes the lower bound for any scehme trying to get rid of the MBE negative encoding by incorporating its effect in he first row, as int eh proposed method.
Proposed method	Generation of the first partial product row and fast 3-bit carry-propagate addition (Fig. 6)	The aim is to reduce the number of partial product rows from [*n*/2] +1 to [*n/2*], thus getting rid of the effect of MBE negative encoding. By having fewer partial product rows he next reduction hardware can be smaller in size and faster in speed. The delay will be higher than the one for the first row for a standard multiplier but it should be lower than any of the other PP rows, thus into inducting any time penalty.
Two's complement	Direct computation of the last partial product row in two's complement performed in parallel with the production of the partial products of the other rows (using the design in Fig.3 and 4 as in [9].	Similarly to the method proposed above this scheme avoids the extra partial product row, and its delay has to be within the delay requirements of the standard PP rows. The above goal is achieved by replacing the partial product generation on the last row with partial product selection of he multiplicand's two's complement, thus eliminating the need for the last *neg* signal.

B. Extension to Rectangular Multipliers

A number of potential extensions to the proposed method exist, including rectangular multipliers, higher radix MBE, and multipliers with fused accumulation [11]. Here, we quickly focus on *m* x *n* rectangular multipliers. With no loss of generality, we assume $m \geq n$, i.e., $m = n + m'$ with $rn' \geq 0$, since it leads to a smaller number of rows; for simplicity, and also with no loss of generality, in the following, we assume that both *m* and *n* are even. Now, we have seen in Fig. 6a, that for $m' = 0$ then the last *neq* bit, i.e., $neg_{n/2-1}$ belongs to the same column as the first row partial product $pp_{n-2.0}$. We observe that the first partial product row has bits up to ; therefore, in order to also include in the first row the contribution of $_{negn/2-1}$ due to the particular nature of operands it is necessary to perform a (m'+3)-bit carry propagation (i.e., a (m' +3)-bit addition) in the sum $qq_{m+1.0qq}m.0\ldots\ qq_{n-2.0} = 00\ldots..\ pp_{n-2,0}$ + 0 1 1….. 0 $neg_{n/2-1.}$ Thus, for rectangular multipliers, the proposed approach can be applied with the cost of a (m' +3)-bit addition. The complete or even partial execution overlap of the first row with other rows generation clearly depends on a number of factors, including the value of m' and the way that the (m' +3)-bit addition is implemented, but still the proposed approach offers an interesting alternative that can possibly be explored for designing and implementing rectangular multipliers.

V. EVALUATION AND RESULTS

In this the proposed method based on the addition of the last *neg* signal to the first row is first evaluated. The designed architecture is then compared with an implementation based on the computation of the two's complement of the last row (referred to as "Two's complement" method) using the designs for the 3-5 decoders, 4-1 multiplexers, and two's complement tree in [9]. Moreover, in the analysis, the standard .MBE implementations for the first and for a generic partial product row are also taken into account (as summarized in Table 2).

For all the implementations, we explicitly evaluate the most common case of a $n < n$ multiplier, although we have shown in Section 4 that the proposed approach can also be extended to *m* x *n* rectangular multipliers. While studying the framework of possible

implementations, we considered the first phase of the multiplication algorithm (i e, the partial product generation) and we focused our attention on the issues of area occupancy and modular design, since it is reasonable to expect that they lead to a possibly small multiplier with regular layout.

The detailed results of some extensive evaluations and comparisons, both based on theoretical analysis and related implementations are reported in [11]. Results encompass the validation by logic synthesis and technology mapping to an industrial cell library. All the results show the feasibility of the proposed approach. Here, for the sake of simplicity, we quickly summarize the results of the theoretical analysis and we check the validity of our estimations through logic synthesis and simulation.

A. High-Level Remarks and Theoretical Analysis

As can be seen from Fig. 6, the generation of the first row is different from the generation of the other rows, basically for two reasons:

- the first row needs to assimilate the last *neg* signal, an operation which requires an addition over the three most significant bit weights;
- the first row can take advantage of a simpler Booth recording, as the y_{-1} bit used by the MBE is always equal to zero (Section 4).

As seen before, in fig. 8. We have a possible implementation to generate the first row, which takes into account the simpler generation of the MBE signals. We have seen that by combining the two parts of Fig. 8 we get Fig. 9, which is faster than Fig. 8, at a possibly slightly larger area cost certainly very marginal with respect to the global area of all the partial product bits coming from the other rows. We have done some rough simulations and found that a good trade-off could be to have the generation of the first bits of the first row carried out by the circuit of Fig. 9, followed by the cascaded addition provided by Fig. 7 (Section 4).

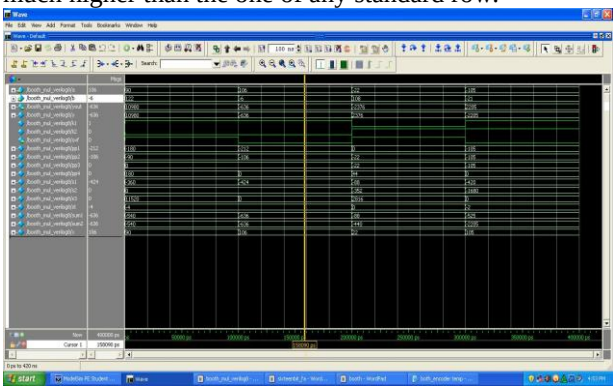
Based on all of the above, our architecture has been designed to perform the following operations:

1. generation of the three most significant bit weights of the first row (through the very small and regular circuitry of Fig 9) and addition to these bits of the *neg* signal (by means of the circuitry of Fig. 7);
2. generation of the other bits of the first row, using the circuitry depicted in Fig. 8; and
3. Generation of the bits of the other rows, using the circuitry of Fig. 1.

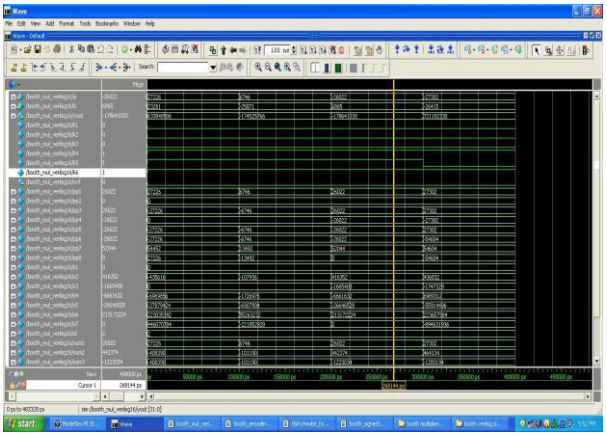
As these three operations can be carried out in parallel, the overall critical path of the proposed architecture emerges from the largest delay among the above paths. We observe that the "Two's Complement" approach has a delay that is longer than the delay to generate the standard partial product rows, becoming even longer as the size *n* of the multiplier increases (e.g., exceeding the delay of a XNOR2 gate starting from *n=16*). On the other hand, according to theoretical estimations, we can see that the delay for generating the first row in the proposed method is estimated to be lower than the delay for generating the standard rows. This means that the extra row is eliminated without any penalty on the overall critical path. With respect to area costs, it can be observed that the proposed method hardly introduces any area overhead with respect to the standard generation of a partial product row. On the other hand, the "Two's Complement" approach requires additional hardware, which increases with the size of the multiplier.

B. Implementation Results

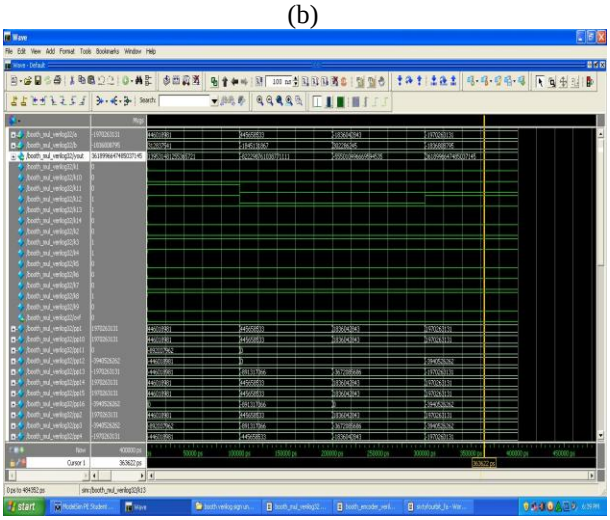
In order to further check the validity of our estimations in an implementation technology, we implemented the designs through logic synthesis and technology mapping to an industrial standard cell library. Specifically, for the logic synthesis, we used Xilinx, XST and the designs were simulated by using modelsim. To perform the evaluation, we obtained the area-delay space for the sole generation of the partial product row of interest (i.e., the first row in the proposed approach, the last row in the implementation presented in [9]). In order to support the comparison, the area-delay space for the generation of the partial product rows using standard MBH implementations was also evaluated, by considering the first row and the other rows of the partial product array separately (Table 2). The results, obtained for n = 8, 16, and 32, are depicted below in Fig. 10. Most importantly, its delay is much higher than the one of any standard row.



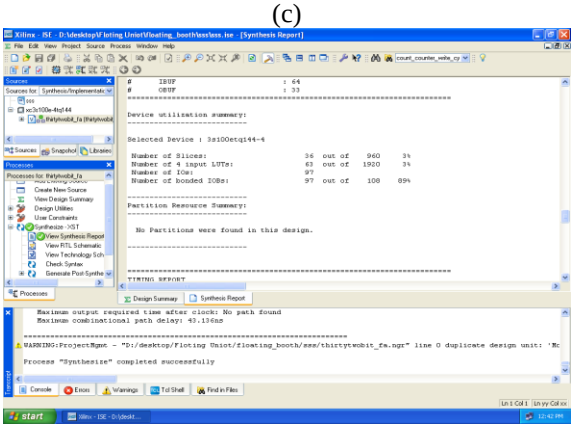
(a)

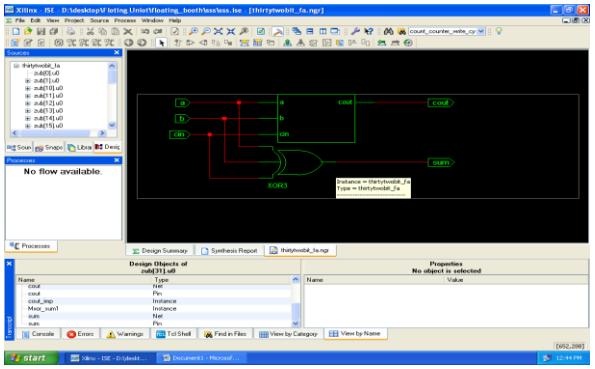


(b)



(c)





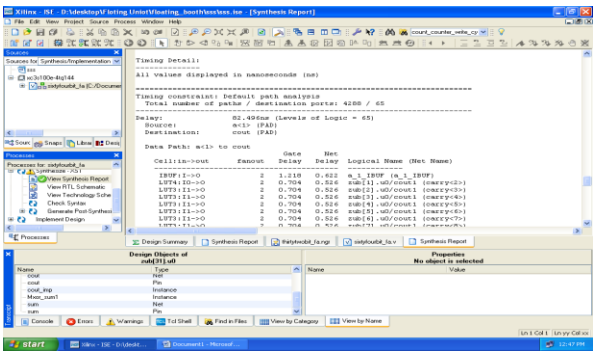


Fig10 Post synthesis results of product generation for different sizes (a) Size of 8. (b) Size of 16. (c) Size of 32

VI. CONCLUSIONS

Two's complement *n x n* multipliers using radix-4 Modified Booth Encoding produce partial products but due to the sign handling, the partial product array has a maximum height of + 1. We presented a scheme that produces a partial product array with a maximum height of, without introducing any extra delay in the partial product generation stage. With the extra hardware of a (short) 3-bit addition, and the simpler generation of the first partial product row, we have been able to achieve a delay for the proposed scheme within the bound of the delay of a standard partial product row generation. The outcome of the above is that the reduction of the maximum height of the partial product array by one unit may simplify the partial product reduction tree, both in terms of delay and regularity of the layout. This is of special interest for all multipliers, and especially for single-cycle short bit-width multipliers for high performance embedded cores, where short bit-width multiplications are common operations. We have also compared our approach with a recent proposal with the same aim, considering results using a widely used industrial synthesis tool and a modern industrial technology library, and concluded that our approach may improve both the performance and area requirements of square multiplier designs. The proposed approach also applies with minor modifications to rectangular and to general radix-B Modified Booth Encoding multipliers.

VII. REFERENCES

1. M.D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann Publishers. 2003.
2. S.K. Hsu, S.K. Mathew. M A Anders, BR. Zeydel. V G. Oklobdzija. R.K. Krishnamurthy. and S.Y. Borkar, "A 110GOPS/ W 16-Bit Multiplier and Reconfigurable PLA Loop in 90-mm CMOS." IEEE J. Solid State Circuits, vol. 41, no. 1, pp. 256-264. Jan. 2006.
3. H Kaul. MA. Anders, S.K. Mathew, S.K Hsu, A. Agarwal, R.K. Krishnamurthy, and S Borkar, "A 300 mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45 nm CMOS," IEEE /. Solid Stat,' Circuits, vol. 45, no. 1, pp 95-101, Jan. 2010.
4. MS Schmookler, M Putrino,, A. Mather, J. Tyler, H.V. Nguyen, C. Roth, M Sharma, M.N'. Pham, and J.Lent, "A Low-Power, High-Speed Implementation of a PowerPC Microprocessor Vector Extension," Proc. 14th IEEE Symp. Computer Arithmetic, pp. 12-19, 1999
5. OL. MacSorlev, "High Speed Arithmetic in Binary Computers," Proc. IRE. vol. 49, pp. 67-91, Jan. 1961.
6. L. Dadda, "Some Schemes for Parallel Multipliers," Alta Freiquenza, vol. 34, pp. 349-356, May 1965.
7. C.S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Trans. Electronic Computers, vol. EC-13, no. 1. pp. 14-17, Feb 1964.
8. D.E. Shaw, "Anton: A Specialized Machine for Millisecond-Scale Molecular Dynamics Simulations of Proteins," Proc. 19th IEEE Symp. Computer Arithmetic, p. 3, 2009.
9. J.-Y. Kang and J-L. Gaudiot "A Simple High-Speed Multiplier Design," IEEE Trims. Computers, vol. 55, no. 10, pp. 1253-1258, Oct. 2006.
10. J.-Y. Kang and J-L. Gaudiot, "A Fast and Well-Structured Multiplier," Proc. Euromicro Symp. Digital System Design, pp. 508 515, Sept. 2004.
11. F. Lamberti, N. Andrikos, C. Antelo, and P. Montuschi, "Speeding-Up Booth Encoded Multipliers by Reducing the Size of Partial Product Array," internal report, http://anth.polito.it/ ir_mbe.pdf, pp. 1-14, 2009
12. E.M. Schwarz, R.M. Averill III, and L.J. Sigal, "A Radix-S CMOS S/390 Multiplier," Proc. 13th IEEE Symp. Computer Arithmetic. pp. 2-9, 1997.
13. W.-C. Yeh and C-W Jen, "High-Speed Booth Encoded Parallel Multiplier Design," IEEE Trans.Computers, vol. 49, no. 7. pp. 692-701, July 2000.
14. Z. Huang and M.D. Ercegovac. "High-Performance Low-Power Left-to-Right Arrav Multiplier Design," EEE Trans. Computers. vol. 54, no. 3, pp. 272-283. Mar. 2005.
15. R. Zimmermann and D.Q. Tran, "Optimized Synthesis ol Sum-of-Products," Proc. Conf. Record of the 37th Asilomar Conf. Signals Systems and Computers, vol. 1, pp. 867-872, 2003.
16. V.G. Oklobdzija. D Villeger, and SS Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Trans Computers, vol. 45, no. 3, pp. 294-306. Mar. 1996
17. P.F. Stelling, C.U. Martel, V.G Oklobdzija, and R. Ravi. "Optimal Circuits for Parallel Multipliers," IEEE Trans Computers vol-47 no. 3, pp. 273-285, Mar. 1998.
18. J. Y. Kang and J.-L. Gaudiot, "A Logarithmic Time Method for Two's Complementation," Proc. int'l Conf. Computational Science, pp 212-219. 2005.
19. K. Hwang, Computer Arithmetic Principles. Architectures, and Design Wiley, 1979
20. R. Hashemian and C.P. Chen, "A New Parallel Technique for Deign of Decrement/Increment and Two's Complement Circuits," Proc. 34th Midwest Sump. Circuits and Systems vol. 2, pp. 887-890, 1991.
21. D. Gajski, Principles of Digital Design Prentice-Hall. 1997.
22. F. Lamberti, N. Andrikos, E. Antelo, and P. Montuschi, "Reducing the computation time in (Short-bit width) two's Complement Multipliers