Reducing Computational Time using Radix-4 in 2's Complement Rectangular Multipliers

Y. Latha
Post Graduate Scholar , Indur institute of Engineering & Technology, Siddipet

K.Padmavathi
Associate. Professor , Indur institute of Engineering & Technology, Siddipet

Abstract - This paper presents a method to spped up Booth encoded multipliers by cing the size of Partial product array using Radix – 4 Modified Booth encoded multiplier . This method is used for higher radices encoding for any size of mxn 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, 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. We evaluated the proposed approach by comparison with some other possible solutions; the results based on a rough theoretical analysis and on logic synthesis showed its efficiency in terms of both area and delay.

Keywords: Multiplication, Radix-4, Modified Booth Encoding, partial product array.

I. INTRODUCTION

In signal processing applications performance strongly depends on the effectiveness of the hardware used for computing multiplications. The high interest in this field is witnessed by the large amount of algorithm and implementations of the multiplication operations. In this short bit width (8-16 bits) two's complement multipliers with single-cycle throughput and latency have emerged to be important building blocks for high performance embedded processors and DSP execution cores. Applications for short bit-width multipliers is the design of SIMD units supporting different data formats. The basic algorithm for multiplication is based three main phases: 1) partial product (PP) genera- tion, 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.

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 nxn bit multiplier using the radix- 4 MBE scheme, generates a PP array with a maximum height of [n/2]+1 rows, each row before the last one being one of the following possible values: all zeros, ±X; ±2X.

The PP reduction is the process of adding all PP rows by using a compression tree [6], [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 addition has the task to sum these two rows and to present the final result in non redundant form i.e., as a single row.

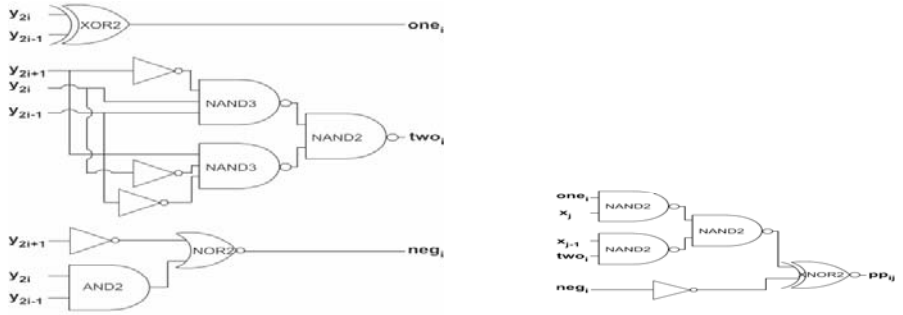
Our aim is to produce a PP array of maximum height of [n/2] to be then reduced by the compressor tree stage. This is the common case of values n which are power of 2, can lead to n implementation where the delay of the compressor tree is reduced by one XOR2 gate.

II. MODIFIED BOOTH ENCODING (RADIX-4)

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	$0 \times X$

The method is to compute the product of a multiplicand X and a multiplier Y, is to produce the partial product array by generating one row for each bit of the multiplier Y. This methodology produces n rows, where n is the size of the multiplier. 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 $-B/2 \times X$ to $B/2 \times X$. Radix-4 is easy to create the multiples of the multiplicand 0; ±X; ±2X. ±2X can be simply obtained by single left shifting of the corresponding terms ±X.



(a) MBE signals generation (b) Partial product generation

Fig. 1. Gate level diagram for partial product generation using MBE (adapted from [8]).

Radix-4 MBE scheme consists of scanning the multiplier operand with a three-bit window and a stride of two bits. 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. For each partial product row, Fig. 1a produces the one, two, and neg signals. These signals are then exploited by the logic in Fig. 1b, along with the appropriate bits of the multiplicand, in order to generate the whole partial product array. The use of radix-4 MBE allows for the (theoretical) reduction of the PP rows to [n/2], with the possibility for each row to host a multiple of $y_i \times X$, with $y_i \in \{0; \pm 1; \pm 2\}$

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 bits. To avoid computing negative encodings, i.e., -X and -2X, the two's complement of the multiplicand is generally used. 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. For 2's complement it requires a neg signal to be added in the LSB position of each partial product row. For nxn multiplier, only [n/2] partial products are generated, the maximum height of the partial product array is [n/2]+1.

When 4-to-2 compressors are used 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 , the extra delay can be further reduced to one XOR2. 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 2, 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 $n = 16$ the maximum column height of the partial product array is 9, with an equivalent delay for the reduction of six XOR2 gates. For a maximum

height of the partial product array of 8, the delay of the reduction tree would be reduced by one XOR2 gate. Alternatively, with a maximum height of 8, it would be possible to use 4 to 2 adders, with a delay of the reduction tree of six XOR2 gates, but with a very regular layout.

III. RELATED WORK

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 is used in order to produce the two's complement by decoding the MBE signals through a 3-5 decoder (Fig. 2a). Finally, a row of 4-1 multiplexers with implicit zero output[1] is used (Fig. 2b) to produce the last partial product row directly in two's complement, without the need for the neg signal. 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. 2

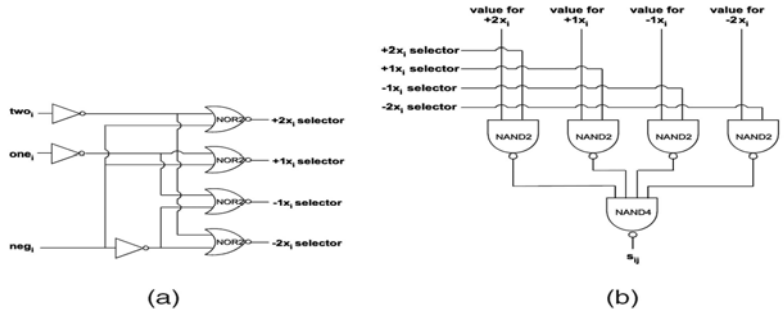


Fig.2. Gate level diagram for the generation of two's complement partial product rows a) 3-5 decoder b) 4-1 multiplexer

IV. BASIC IDEA

4.1 Square Multipliers

The high level description of the proposed idea is as follows:

1. generation of most significant three 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.

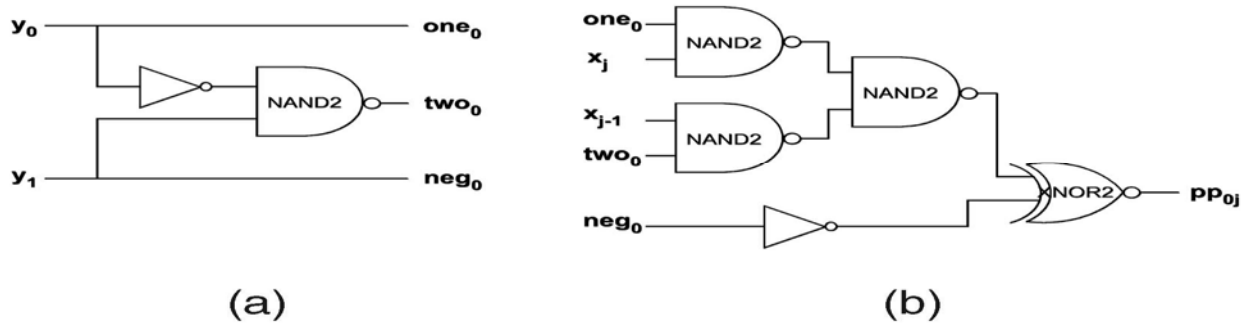


Fig. 3. Gate-level diagram for first row partial product generation. (a) MBE signals generation. (b) Partial product Generation.

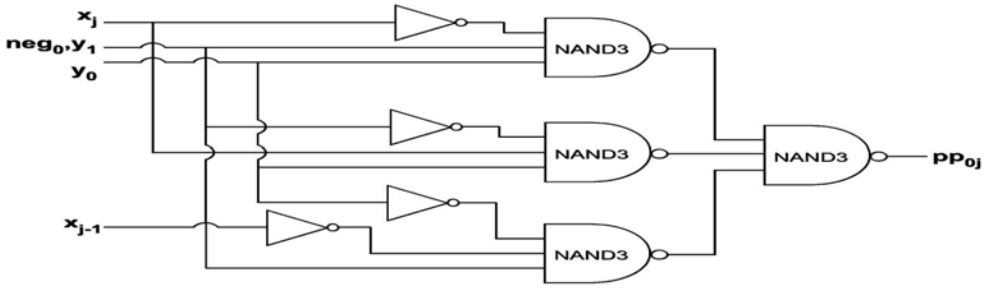


Fig. 4. Combined MBE signals and partial product generation for the first row (improved for speed).

								x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
							×	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
					$\overline{pp_{80}}$	pp_{80}	pp_{80}	pp_{70}	pp_{60}	pp_{50}	pp_{40}	pp_{30}	pp_{20}	pp_{10}	pp_{00}
			1		$\overline{pp_{81}}$	pp_{71}	pp_{61}	pp_{51}	pp_{41}	pp_{31}	pp_{21}	pp_{11}	pp_{01}		neg_0
		1		$\overline{pp_{82}}$	pp_{72}	pp_{62}	pp_{52}	pp_{42}	pp_{32}	pp_{22}	pp_{12}	pp_{02}		neg_1	
s_9	s_8	s_7		s_6	s_5	s_4	s_3	s_2	s_1	s_0		neg_2			

Fig. 5. Partial product array by applying the two's complement computation method in [7] to the last row.

										x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
								×	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	
							$\overline{pp_{80}}$	pp_{70}	pp_{60}	pp_{50}	pp_{40}	pp_{30}	pp_{20}	pp_{10}	pp_{00}		
						1	1		neg_3								
				1	$\overline{pp_{81}}$	pp_{71}	pp_{61}	pp_{51}	pp_{41}	pp_{31}	pp_{21}	pp_{11}	pp_{01}		neg_0		
			1	$\overline{pp_{82}}$	pp_{72}	pp_{62}	pp_{52}	pp_{42}	pp_{32}	pp_{22}	pp_{12}	pp_{02}		neg_1			
1	$\overline{pp_{83}}$	pp_{73}	pp_{63}	pp_{53}	pp_{43}	pp_{33}	pp_{23}	pp_{13}	pp_{03}		neg_2						

(a) Basic idea

								x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
							×	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
					$\overline{qq_{90}}$	qq_{90}	qq_{80}	qq_{70}	qq_{60}	pp_{50}	pp_{40}	pp_{30}	pp_{20}	pp_{10}	pp_{00}
			1		$\overline{pp_{81}}$	pp_{71}	pp_{61}	pp_{51}	pp_{41}	pp_{31}	pp_{21}	pp_{11}	pp_{01}		neg_0
		1		$\overline{pp_{82}}$	pp_{72}	pp_{62}	pp_{52}	pp_{42}	pp_{32}	pp_{22}	pp_{12}	pp_{02}		neg_1	
1	$\overline{pp_{83}}$	pp_{73}	pp_{63}	pp_{53}	pp_{43}	pp_{33}	pp_{23}	pp_{13}	pp_{03}		neg_2				

(b) Resulting array

Fig. 6. Partial product array after adding the last *neg* bit to the first row.

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 par- tial product array (Fig. 1).	The delay to generate a generic partial product row (other than the first one) in a standard $n \times n$ multiplier represents the upper bound for any design aimed at removing the last neg signal
Standard multiplier (First row)	Revisited implementation of the MBE for the first row: logic for the generation of the partial product row is simplified as the $y-1$ bit is always equal to zero	The delay to produce the first row constitutes the lower bound for any scheme trying to get rid of the MBE negative encoding by incorporating its effect in the shortest path (i.e., in the first row, as in the proposed method).
Proposed method	Generation of the first partial product row and fast 3-bit carry propagate addition	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, the next reduction hardware can be smaller in size and faster in speed. The delay will be higher than the one of the first row for a standard multiplier, but it should be lower than any of the other PP rows, thus not inducing any time penalty.
Two's complement	Direct computation of the last partial product row in two's complement per- formed in parallel to the production of the partial products of the other rows	Similarily as the proposed method, also this scheme avoids the extra partial product row, and its delay has to be within the delay requirements of the other PP rows. The above goal is achieved by replacing partial product generation on the last row with partial product selection of the multiplicand's two's comple- ment, thus eliminating the need for the last *neg* signal. With respect to similar techniques

Table 2 DESIGNS FOR THE GENERATION OF THE PARTIAL PRODUCT ROWS CONSIDERED IN THE EVALUATION

4.2 *Rectangular Multipliers:*

- $m \times n$ rectangular multiplier;
- higher radix Modified Booth Encoding (e.g. radix-8);
- multipliers with fused accumulation.

With no loss of generality we assume $m \geq n$, i.e. $\overline{m} = n + m^0$ with $m^0 \geq 0$, since it leads to the 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. 6(a) that for $m^0 = 0$, then the last *neg* 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 $pp_{m,0}$, and therefore, in order to include in the first row also the contribution of $neg_{n/2-1}$, due to the particular nature of operands it is necessary to perform a $(m^0 + 3)$-bit carry propagation (i.e., a $(m^0 + 3)$-bit addition) in the sum $qq_{m+1,0}\, qq_{m+1,0}\, qq_{m,0} \ldots qq_{n-2,0} = 0\ 0\ pp_{m,0} \ldots pp_{n-2,0} + 0\ 1\ 1 \ldots 0\ neg_{n/2-1}$. Therefore, for rectangular

multipliers, the proposed approach can be applied with the cost of a $(m^{0} + 3)$-bit addition.

Although we have explicitly focused our attention to radix-4 MBE, the proposed method can be easily extended to any radix-B MBE. It is easily observed, by redrawing the equivalent of Fig. 6(a) for another radix-B MBE, that the *neg* signal of the last row can be included in the first row by using a simple 3-bit adder for a $n \times n$ multiplier and by using a $(m^{0} + 3)$-bit adder for the more general case of a $(n + m^{0}) \times n$ adder. The use of a combined multiplier with accumulation is also common. In this case, the proposed approach can also be used and does not loose the benefit to reduce by one the number of rows to be added. Although this reduction does not necessarily lead to a reduction in the computation time for some values of n, it still remains interesting and useful since it is very reasonable to expect some savings and more regularity in the compression tree. We have not evaluated such potential reductions, because their impact could be strongly dependent on the value of n and on the strategy which has been identified to design the compression tree.

V. CONCLUSIONS

Two's complement $n \times n$ bit multipliers using $\frac{n}{2}$radix-4 Modified Booth Encoding$_2$produce *[n/2]* partial products but due the sign handling, the partial product array has a maximum height of *[n/2]+1*. We presented a scheme that produces a partial product array with a$_2$ maximum height of [n/2], without introducing extra delay in the partial product generation stage for m x n bit multipliers. With the extra hardware of a (short) 3-bit addition, and the simpler generation of the first partial product, we have been allowed to achieve a delay for the proposed scheme within the bound of the delay of a standard partial product 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, in terms of delay and regularity of the layout. This is of special interest for all multipliers but especially for short bit-width multipliers for high performance embedded cores, where short but fast bit-width multiplications could be common operations.

REFERENCES

[1] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Los Altos, CA, USA: Morgan Kaufmann Publishers, 2003.

[2] S. K. Hsu, S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Ok- lobdzija, R. K. Krishnamurthy, and S. Y. Borkar, "A 110gops/w 16-bit multiplier and reconfigurable pla loop in 90-nm cmos," *IEEE Journal of Solid State Circuits*, vol. 41, pp. 256–264, Jan. 2006.

[3] M. S. 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(tm) microprocessor vector extension," *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, p. 12, 1999.

[4] A. D. Booth, "A signed multiplication technique," *Quarterly J. Mech. Appl. Math.*, vol. 4, 1951.

[5] L. Dadda, "Some schemes for parallel multipliers," *Alta Fre- quenza*, vol. 34, May 1965.

[6] O. L. MacSorley, "High speed arithmetic in binary computers," *Proceedings IRE*, vol. 49, pp. 67–91, Jan. 1961.

[7] J.-Y. Kang and J.-L. Gaudiot, "A simple high-speed multiplier design," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1253–1258, Oct. 2006.

[8] "A fast and well-structured multiplier," *Proceedings of Euromicro Symposium on Digital System Design*, pp. 508– 515, Sept. 2004.

[9] E. M. Schwarz, R. M. A. III, and L. J. Sigal, "A radix-8 cmos s/390 multiplier," *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, p. 2, 1997.

[10] F. Lamberti, N. Andrikos, E. Antelo, and P. Montuschi, "Speeding-Up Booth Encoded Multipliers by Reducing the Size of Partial Product Array," internal report, http://arith.polito.it/ ir_mbe.pdf, pp. 1-14, 2009.

[11] F. Lamberti, N. Andrikos, E. Antelo, and P. Montuschi, "Reducing the computation time in (Short-bit width) Two's Complement Multipliers