MINIMIZING ENERGY DISSIPATION IN HIGH-SPEED MULTIPLIERS

Rafael Fried

Swiss Federal Institute of Technology (EPFL), Electronics Laboratory, ELB-Ecublens, CH-1015 Lausanne, Switzerland
Phone: +41 21 693 39 76, Fax: +41 21 693 36 40, E-mail: rafif@leghp10.epfl.ch

ABSTRACT

This paper presents a new two-gate-delay implementation of the Booth encoder and partial product generator, which eliminates the unnecessary glitches associated with the Booth multiplier. In addition, a modified signed/unsigned (MSU) and modified sign-generate (MSG) algorithms, suitable especially for signed/unsigned multipliers, were developed in order to reduce the compression level needed in the Wallace tree, and hence reduce the multiplier hardware.
Using these features reduces the multiplier array energy dissipation by about 30% and increases speed by about 10%.

1. INTRODUCTION

Currently, it become imperative for reduced instruction set computers (RISC) and digital signal processors (DSP) to use less energy without sacrificing their computation throughput. Hence, the parallel multiplier as one of the key building blocks of RISC and DSP, must address simultaneously the low-power and high-speed design issues.
In general there are two basic approaches to enhance the speed of parallel multipliers, one is the Booth algorithm and the other is the Wallace tree compressors or counters. However, both typically lead to excessive energy dissipation [1].
When only Wallace tree is used to compress the number of partial products [2], the multiplier array becomes very large due to the large number of gates and the interconnect wires. This leads to high energy dissipation. On the other hand, when the Booth algorithm is used [3][4][5], a lot of unnecessary glitches occur in the multiplier array as a result of the race condition between the multiplicand and the multiplier, due to the Booth encoder and the partial product generator. This again leads to high energy dissipation. Furthermore, when not optimizing the Booth algorithm, to match the special conditions of the array, both in terms of operands sizes and in terms of sign extension bits (as for example when signed/unsigned multipliers are used), the result is a large increase in the number of full adders (FA) needed for the Wallace tree compressors.

In this work a new implementation of the Booth encoder and partial product generator is presented, that eliminates the unnecessary glitches associated with the modified Booth algorithm. In addition, it exhibit a very short delay time, only two gates, from the input operands to the partial products. Combining this Booth encoder with the MSG algorithm, the MSU algorithm, and the 4-2 based Wallace tree compressors [3][4][6], leads to the fastest possible multiplier array with reduced energy dissipation.

2. THE MSU ALGORITHM

The operands of the multiplier presented here have two basic operation modes: 16-bit unsigned numbers and 16-bit signed numbers, as shown in Fig. 1, where **SE** is the sign extension bit. The additional bit (17^{th}) is needed to represent the operands in both modes in two's complement.

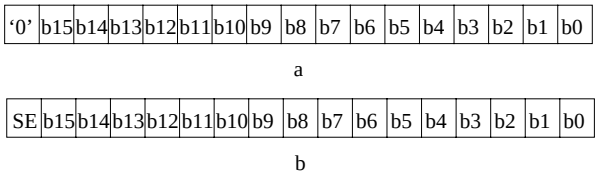

Fig. 1 *Signed and unsigned 16-bit operands:*
a. Unsigned operand
b. Signed operand

The range of the unsigned operands is $0 \le X, Y \le 2^{16} - 1$ and of the signed operands $-2^{15} \le X, Y \le 2^{15} - 1$. The result is in the range $-2^{31} + 2^{15} \le S \le 2^{32} - 2^{17} + 1$, and it can be represented in two's complement using 33 bits.
In general, the modified Booth algorithm is applicable only for two's complement operands. The bit-pair Booth algorithm is based on partitioning the multiplier into overlapping groups of 3 bits. Each group is then encoded to generate a correct partial product. The n-bit multiplier Y is written in two's complement as:

$$Y = -y_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{i=n-2} y_j \cdot 2^i \qquad (1)$$

It can be rewritten as:

$$Y = \sum_{i=0}^{i=n/2-1} (y_{2i-1} + y_{2i} - 2 \cdot y_{2i+1}) \cdot 2^{2i} \qquad (2)$$
$$with \qquad y_{-1} = 0$$

where the term in brackets, in (2), has values in the set $\{-2, -1, 0, 1, 2\}$. Each recoded value performs a certain operation on the multiplicand *X* (and accordingly adds *'0'* or *'1'* to the LSB) as illustrated in Table I.

Table I: *Modified Booth algorithm: recoding scheme.*

y_{2i+1}	y_{2i}	y_{2i-1}	operation on ***X***	**ADD** to LSB
0	0	0	$0 \cdot X$	0
0	0	1	$+1 \cdot X$	0
0	1	0	$+1 \cdot X$	0
0	1	1	$+2 \cdot X$	0
1	0	0	$-2 \cdot X$	+1
1	0	1	$-1 \cdot X$	+1
1	1	0	$-1 \cdot X$	+1
1	1	1	$0 \cdot X$	0

This algorithm is suitable only for operands with even number of bits. Hence, in our case an additional bit is needed for *Y*. The result is a 17 x 18-b multiplier array (*X* has 17 bits and *Y* has 18 bits). In this case a maximal number of 10 bits can be added in the same bit position (column number 16).

In order to achieve the fastest multiply operation, using the Wallace tree compression, two rows of 4-2 compressors and one row of 6-2 compressors should be used, resulting in an equivalent delay of 8 XOR gates [6].
A modified signed/unsigned Booth algorithm is proposed next in order to reduce the number of rows in the array, and hence reduce hardware and increases speed. Since the operand *Y* in the signed mode can be represented by using only 16-bits, a solution must be given for the unsigned operand mode. In this case (2) can be rewritten as:

$$Y = \sum_{i=0}^{i=n/2-2} (y_{2i-1} + y_{2i} - 2 \cdot y_{2i+1}) \cdot 2^{2i} \qquad (3)$$
$$+ (y_{n-3} + y_{n-2} + 2 \cdot y_{n-1}) \cdot 2^{n-2}$$

The last term in brackets, in (3), has values in the set $\{0, 1, 2, 3, 4\}$, and must be separated into two terms:

$$Y = \sum_{i=0}^{i=n/2-2} (y_{2i-1} + y_{2i} - 2 \cdot y_{2i+1}) \cdot 2^{2i} \qquad (4)$$
$$+ (y_{n-3} + y_{n-2}) \cdot 2^{n-2} + y_{n-1} \cdot 2^{n-1}$$

Eq. (4) is the modified signed/unsigned version of the Booth algorithm. Using it enables to have a 17 x 16-b multiplier array.

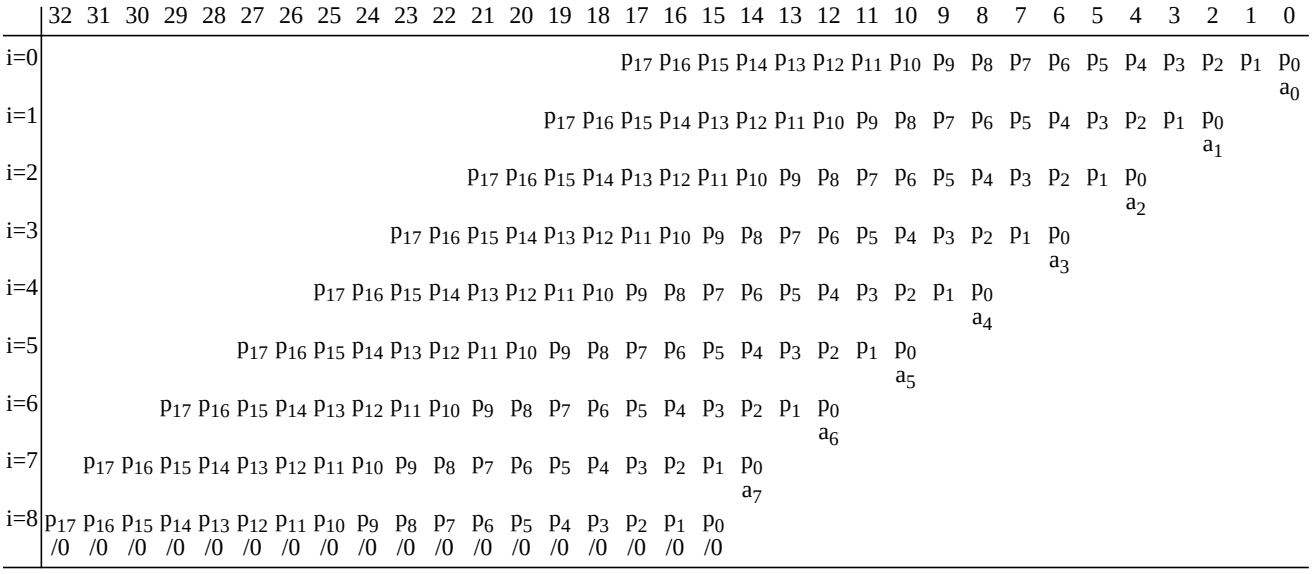

Fig. 2: *17 x 16-b multiplier array after the modified Booth partial product compression.*

When working in the signed mode the "regular" Booth algorithm (2) is used. In the unsigned mode, bit y_{n-1} in the last Booth encoder (y_{15} in the case of 16-bit) should be replaced by *'0'*, this satisfies the term $(y_{n-3}+y_{n-2})\cdot 2^{n-2}$ in (4). An additional partial product should be generated according to the real value of y_{n-1}, in order to satisfy the last term in (4) $y_{n-1}\cdot 2^{n-1}$.

The multiplier array, after using the MSU algorithm is depicted in Fig. 2. The p^i_j are the partial products, and a_i are the bits added to the LSB. The first seven rows *(i = 0 - 6)* are yielded from the "regular" Booth algorithm. The partial products in the eighth row *(i = 7)* are yielded either from the "regular" term given in (2), or the modified second term given in (4). The last partial product *(i = 8)* is either all zeroes, when using the signed mode, or the partial product given by the AND function between y_{15} and the *X* operand.

In this array maximum of only 9 bits are added in the same bit position. Hence, in order to achieve the fastest multiplier operation, only one row of 9-2 compressors should be used, resulting in a delay equivalent to only 7 XOR gates [6].

3. MSG ALGORITHM

In some cases, when Wallace tree compression is used, the partial products are signed extended till the MSB of the array. The result is a large power and area waste due to the unnecessary compressors used for the extended sign bits.

In order to reduce the array to a rectangle two basic sign extension methods are typically used, namely the sign-propagate and the sign-generate [1][7]. The sign-propagate algorithm is useless in fast multiplier arrays due to its large delay time as a result of the series dependence of the sign extension of each partial product on that of the previous one.

According to the sign-generate algorithm [7] the result of adding all the sign extension bits of a 17 x 16-b multiplier can be written as:

$$Sgn = \left(s_0\cdot\sum_{i=17}^{32}2^i\right)\cdot 2^0+\left(s_1\cdot\sum_{i=17}^{30}2^i\right)\cdot 2^2+ \left(s_2\cdot\sum_{i=17}^{28}2^i\right)\cdot 2^4+\left(s_3\cdot\sum_{i=17}^{26}2^i\right)\cdot 2^6+ \left(s_4\cdot\sum_{i=17}^{24}2^i\right)\cdot 2^8+\left(s_5\cdot\sum_{i=17}^{22}2^i\right)\cdot 2^{10}+ \left(s_6\cdot\sum_{i=17}^{20}2^i\right)\cdot 2^{12}+\left(s_7\cdot\sum_{i=17}^{18}2^i\right)\cdot 2^{14} \quad (5)$$

where s_i is the sign bit of the partial product in the i^{th} row.

Using the two equivalences:

$$\sum_{i=j}^{k}2^i = 2^{k+1}-2^j \quad (6)$$

$$s_i = 1-\bar{s}_i \quad (7)$$

Sgn becomes:

$$Sgn = \left[7-\sum_{j=0}^{7}\bar{s}_j\right]\cdot 2^{33}+\sum_{j=0}^{7}\bar{s}_j\cdot 2^{17+2j} + 2^{17}+2^{18}+2^{20}+2^{22}+2^{24}+2^{26}+2^{28}+2^{30}+2^{32} \quad (8)$$

The first term in (8) affects only the bits 34 and above and can be omitted. When using (8) all the sign extension bits, of all partial products, can be replaced by the following steps:

- Inverting the MSB of all the partial product (p_{17})
- Adding *'1'* to the left of each partial product
- Adding *'1'* in bit column number 17

Although this algorithm reduces significantly the number of unnecessary FA used to compress the sign extension bits when signed operands are used, the total result is a very small reduction in the overall number of FA needed for the multiplier array when unsigned operands are also used. The reason for that is the increase in the number of bits in column number *n+1* (column 17 in the 16-bit multiplier, depicted in Fig. 2, will have 10 bits).

In order to achieve the most economical array a modified sign-generate algorithm is presented hereafter.

To eliminate the *'1'* in column number 17, (8) is rewritten as:

$$Sgn = \sum_{j=1}^{7}\bar{s}_j\cdot 2^{17+2j} + 2^{20}+2^{22}+2^{24}+2^{26}+2^{28}+2^{30}+2^{32} + 2^{17}\cdot(3+\bar{s}_0) \quad (9)$$

where the first term of (8) was omitted.

The last term in (9) has the value 3 or 4, depending on the value of s_0. When $\bar{s}_0$ = *'0'* the last term is *'011'* or '$\bar{s}_0\, s_0\, s_0$' and when $\bar{s}_0$ = *'1'* this term equals *'100'* or again '$\bar{s}_0\, s_0\, s_0$'. Fig. 3 depicts the final optimized multiplier array with both modified signed/unsigned Booth algorithm and the modified sign-generate algorithm. A maximum number of 9 bits is added in a column (columns 14-19).

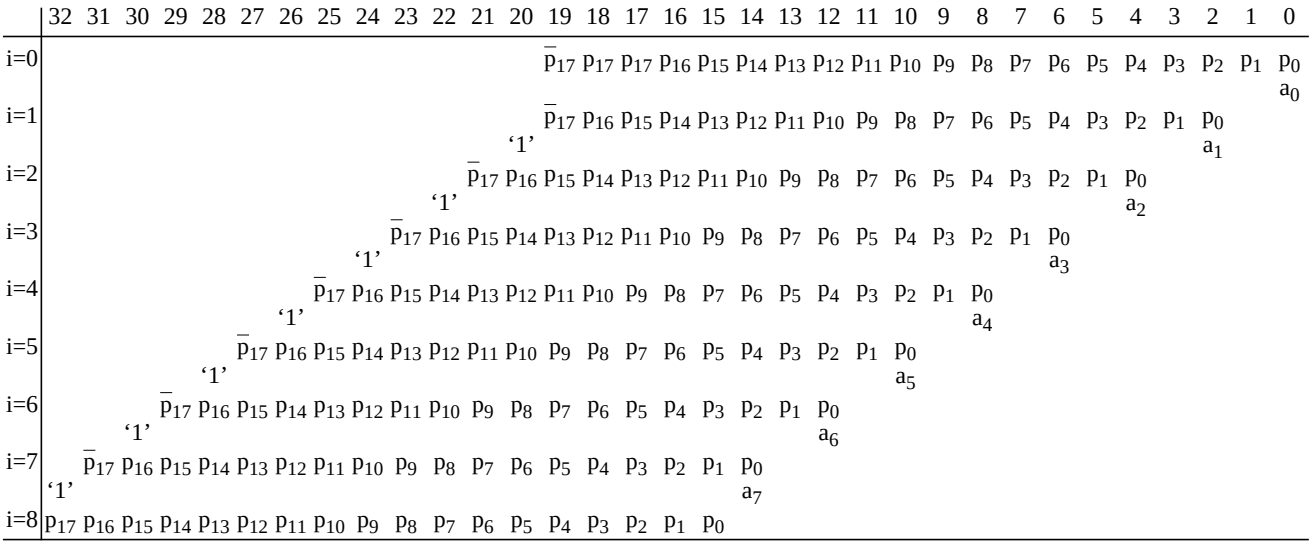

Fig. 3: *17 x 16-b optimized multiplier array with modified sign-generation.*

To compress this array only one row of 9-2 compressors is needed. This results in a delay time equivalent to 7 XOR gates, with a total number of 170 FA. On the other hand when not using the MSU and MSG the array compression time is equivalent to 8 XOR gates and 210 FA are used.
The final adder is a 27-bit fast adder for the higher bits [8]-[10], while the lower 7 bits of the final result can be computed simultaneously to the array compression.

4. BOOTH ENCODER AND PARTIAL PRODUCT GENERATOR

The implementation of the modified Booth algorithm, is typically a major cause of energy dissipation, due to the race condition between the *X* and *Y* operands.
The most common implementation of the Booth encoder and the partial product generator is depicted in Table II and Fig. 4.

Table II: *Standard encoding of Booth algorithm.*

y_{2i+1}	y_{2i}	y_{2i-1}	P1	P2	Z	M1	M2
0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	1	0	0	0
1	0	0	0	0	0	0	1
1	0	1	0	0	0	1	0
1	1	0	0	0	0	1	0
1	1	1	0	0	1	0	0

This implementation has large encoders and small partial product generators (only 8 transistors), resulting is a small

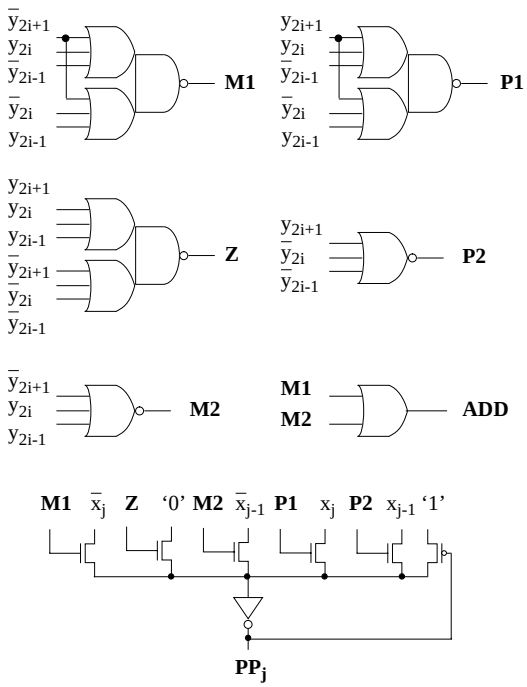

Fig. 4 *Standard encoding of Booth algorithm.*

array area. This is due to the fact that in a n x n multiplier the partial product generator is placed $(n/2)\cdot(n+1)$ times while the encoder only $(n/2)$ times. Furthermore, the encoder's load is comparatively small since only one NMOS is used for each column in each row, for each encoder's control line.
A more compact implementation is presented in Table III and Fig. 5 [4][5]. In this case more transistors are used for the partial product generator, but on the other hand the encoder is much simpler, and only three control lines are

passed for each row. A similar implementation that is used to optimize the dimensions of the array slice can be found in [6].

The drawback of all these implementations is the unnecessary glitches caused on the partial product. This problem is best demonstrated by an example using the implementation presented in Fig. 5.

Table III: *Compact encoding of Booth algorithm.*

y_{2i+1}	y_{2i}	y_{2i-1}	X1	X2	NEG
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	0	0	1

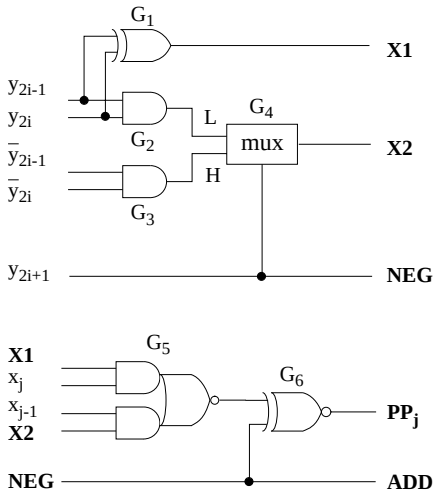

Fig. 5 *Compact encoding of Booth algorithm.*

Assuming the multiplier is a 4 x 4-b, the current operands are $X = Y = 0101$ and they are changed to $X = 1111$ and $Y = 1001$. In the current cycle the two 3-bit sub-strings of the multiplier Y are:

$$\{y_{2i+1}, y_{2i}, y_{2i-1}\} = \{0, 1, 0\} \quad (10)$$

According to Table III the encoded bits are: **X1** = *'1'* and **X2** = **NEG** = *'0'*. Hence, the fourth partial product in the second column ($i = 1, j = 3$) is $\mathbf{PP^1_3}$ = *'0'*. In the next cycle the operands are changed. The left sub-string *(i = 1)* is:

$$\{y_{2i+1}, y_{2i}, y_{2i-1}\} = \{1, 0, 0\} \quad (11)$$

and the new encoded bits are **X1** = *'0'* and **X2** = **NEG** = *'1'*. It is seen from Fig. 5, that the **NEG** signal propagates to the partial product generator without any gate delay, and hence the value of the new partial product, after one gate delay $T = T_{DG6}$ will be $\mathbf{PP^1_3}$ = *'1'*. After additional gate delay $T = T_{DG6} + T_{DG5}$, the partial product will be inverted, $\mathbf{PP^1_3}$ = *'0'*,due to the change of the operand X ($x_3 = x_2 =$ *'1'*). The third time the partial product will be inverted, $\mathbf{PP^1_3}$ = *'1'*, is after additional one gate delay $T = T_{DG6} + T_{DG5} + T_{DG1}$, due to the change of the encoded signal **X1.** Finally, the partial product will gain its steady value, $\mathbf{PP^1_3}$ = *'0'*, at $T = T_{DG6} + T_{DG5} + T_{DG1} + T_{DG4}$, due to the change of **X2**. For this analysis it was assumed that all gates have approximately the same propagation delay time. This is especially true if pass-logic families are used.

It should be mentioned that any change in the value of the partial products causes also a change all along the multiplier array, and the final adder. Thus, in the example presented above some parts of the array will exhibit four logic state changes when no change was actually needed at all. This energy dissipation associated with the glitches in the modified Booth algorithm is an important portion of the total energy dissipation of the whole multiplier [1].

The problem of spurious transitions is not unique for the implementation presented in Fig. 5, it can be verified that it appears in all the previously reported Booth encoders.

Two basic approaches can be used in order to eliminate the unnecessary glitches in the Booth algorithm. One is to latch all the partial products and allow them to change only after steady-state was reached in the encoder and the partial product generator. This can be done by using a clock derivative from the global clock, whose duty cycle is defined according to the slowest path in the Booth implementation. However, this approach requires large area and dissipates a lot of energy by itself.

Table IV: *Race-free encoding of Booth algorithm.*

y_{2i+1}	y_{2i}	y_{2i-1}	NEG	X1	X2P	ZP
0	0	0	0	0	1	1
0	0	1	0	1	0	1
0	1	0	0	1	0	0
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	0	1	1	1	0	0
1	1	0	1	1	0	1
1	1	1	1	0	1	1

The second approach is to synchronize all the path in the encoder and the partial product generator. It can be implemented by using a different recoding scheme as presented

in Table IV and Fig. 6. The principle here is to achieve the fastest possible equal path for all signals emerging from the *X* and *Y* operand latches. It can be verified that the unnecessary glitch problem demonstrated above does not occur in this implementation.

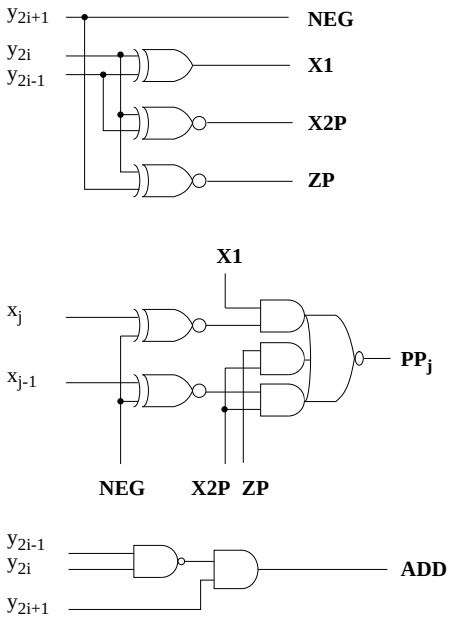

Fig. 6 *Race-free encoding of Booth algorithm.*

The following properties should be noted:

- The circuit uses only XOR and NXOR gates till the last stage of the partial product generator. It means that all the path can be equalized to have exactly the same propagation delay.
- The delay from *X* and *Y* to **PP** is only two gates, one XOR/NXOR and the output complex gate.
- The encoder is very compact.
- Four control lines are used for each row.
- The load, for each column in each row, on **X1** and **X2P** is one gate, and **NEG** is loaded with two gates. This is the same as in the compact encoder. The additional control line **ZP** is loaded with one gate.
- Complementary inputs are not needed.

The penalty for this fast and race-free implementation is the larger area used for the partial product generators [11]. The full CMOS implementation of the partial product generator consists of 24 transistors, compared to only 15 that are needed for the compact implementation presented in Fig. 5. As a result, the area that is spared by reducing the number of FA that are needed to compress the array, is used for the larger partial product generators.

5. SUMMARY

Glitch-free Booth encoder and partial product generator were presented, together with two algorithms: MSU and MSG. Using all those features yields on one hand the fastest signed/unsigned multiplier array, while on the other hand decreases significantly its energy dissipation.

The speed enhancement is due to the reduction of the compression level in the multiplier array. Without using the MSG and MSU algorithms the compression time is equivalent to 8 XOR gates (3 at the 4-2 first compression level and 5 more at the 6-2 second compression level). When using the MSG and MSU only one compression level 9-2 is used, with a delay equivalent to 7 XOR gates. Energy dissipation is reduced due to the elimination of the glitches associated with the Booth algorithm, and the reduction of the number of FA needed to compress the multiplier array.

6. REFERENCES

[1] A. Bellaouar and M. I. Elmasry, *"Low-Power Digital VLSI Design circuits and systems", Kluwer Academic Publishers*, Boston 1995.

[2] K. Yano *et al.*, "A 3.8 nS CMOS 16 x 16-b Multiplier Using Complementary Pass-Transistor Logic", IEEE JSSC, Vol. 25, No. 2, pp. 388-395, Apr. 1990.

[3] J. Mori *al.*, "A 10 nS 54 x 54-b CMOS Parallel Structured Full Array Multiplier with 0.5-μm CMOS Technology", IEEE JSSC, Vol. 26, No. 4, pp. 600-605, Apr. 1991.

[4] N. Ohkubo *et al.*, "A 4.4 nS CMOS 54 x 54-b Multiplier Using Pass-Transistor Multiplexer", IEEE JSSC, Vol. 30, No. 3, pp. 251-257, Mar. 1995.

[5] G. Goto *et al.*, "A 54 x 54-b Regularly Structured Tree Multiplier", IEEE JSSC, Vol. 27, No. 9, pp. 1229-1235, Sept. 1992.

[6] P. J. Song and G. De Micheli, "Circuit and Architecture Trade-offs for High-Speed Multiplication", IEEE JSSC, Vol. 26, No. 9, pp. 1184-1198, Sept. 1991.

[7] M. Annaratone, *"Digital CMOS circuit Design", Kluwer Academic Publishers*, Boston 1986.

[8] H. Lindkvist and P. Andersson, "Techniques for Fast CMOS Based Conditional sum Adders", Int. conf. Comp. Design, pp.626-635, Oct. 1994.

[9] T. K. Callaway and E. E. Swartzlander Jr., "Estimating the Power consumption of CMOS Adders", 11th Symp. on Comp. Arithmetic, pp. 210-219, June 1993.

[10] M. Suzuki *et al.*, "A 1.5 nS CMOS ALU in Double Pass-Transistor Logic", IEEE JSSC, Vol. 28, No. 11, pp. 1145-1151, Nov. 1993.

[11] A. Inoue et al., "A 4.1nS Compact 54x54b Multiplier Utilizing Sign Select Booth encoders", ISSCC'97 pp. 416-417.