Design of Fault-Secure Parity-Prediction Booth Multipliers

M. NICOLAIDIS, R.O. DUARTE
TIMA Laboratory, Reliable Integrated Systems Group
France

ABSTRACT: The basic drawback of parity prediction arithmetic operators is that they may not be fault secure for single faults. In a recent work we have proposed a theory for achieving fault secure design for parity prediction multipliers and dividers. This paper has not considered the case of Booth multipliers using operand recoding. This case is analyzed here. Parity prediction logic and fault secure implementation for this scheme is derived.
Keywords: Self-checking circuits, Booth multipliers

I. INTRODUCTION: Since arithmetic units (i.e. adders, ALUs, multipliers and dividers) are essential elements of computers, designing efficient self-checking arithmetic units is mandatory for designing self-checking and fault tolerant computers. Early schemes for self-checking arithmetic units were based on arithmetic residue codes [AVI73]. The parity prediction scheme was also proposed for the same purposes [SEL68]. The basic drawback of this scheme is that it may not achieve the fault secure property because single faults propagate on output errors of random multiplicity which are undetectable by the parity code.

[NIC 97] proposed a design technique allowing to achieve the fault secure property in parity prediction multipliers and dividers, by constraining propagation of single faults into output errors of odd multiplicity. This work considers multipliers that do not use operand recoding. Since Booth multipliers are among the most popular multiplier schemes, we are extending our previous work in the case of these schemes. The proposed solutions are implemented into a macroblock generator and integrated into our framework of CAD tools for data path design [DUA 97]. These tools today include macroblock generators for various self-checking adders and ALUs, shifters for single and multiple position shifts, register files, dividers, parity prediction multipliers, arithmetic code based multipliers, as well as the related parity, double-rail, and arithmetic code self-checking checkers.

II. Multipliers with a Recoded Operand

In 1951. A.D. Booth presented a signed binary multiplication technique that is used nowadays in a large number of multiplier structures [Mor91] [Sat91] [Bur94] [Yu95] [Mak96]. The Booth algorithm reduces the number of partial products by recoding the multiplier (A). As it was originally proposed, the Booth algorithm performs the recoding serially [Boo51]. Therefore, the Modified Booth Algorithm - Booth2 [McS61] which performs the recoding in parallel is used. Booth2 provides a reduction of partial products from n2 to n x (n+1)/2 (a decreasing of the number of dots to be added in the dot diagram). This reduction however is not a complete saving, since the partial product selection circuit is more complex than a single AND gate. In Booth2 algorithm, the multiplier (A) is partitioned into overlapping groups of 3 bits. Each of these groups is decoded in parallel to select a single partial product according to table 1. In table 2 is presented the relationship between the partial product and the recoded signed bits and table 3 gives the truth table for partial products.

Figure 1 shows an 6x6 Booth2 signed multiplication example. Note that the input operands (A and B) and the result (R) are in a signed two complement notation. The same topology (hardware structure) can be used to perform non signed multiplications. Figure 2 presents the non-signed multiplication. Note that, to perform non-signed multiplication an extra bit is necessary for the multiplier and for the multiplicand (see figure 2).

There are also, Booth3 and Booth4 multipliers that were further proposed but they do not provide significantly better results than Booth2 due to the complexity of the decoding and selection circuit and the irregularity of routing for diverse topologies [Twa95]. In this work will be considered only Booth2 recoding implementations.

Table 1- Partial Product Selection Table

Multiplier Bits (A)	Selection
000	+0
001	+B
010	+B
011	+2B
100	-2B
101	-B
110	-B
111	+0

Table 3 - Truth Table for Partial Products

Multiplier Recoded Digit	Output		add1
0 ('0')	ppi = 0	for i=1, ..., 8	0
+1 ('p1')	ppi = bi	for i=1, ..., 8	0
-1 ('m1')	ppi = ¬bi	for i=1, ..., 8	1
+2 ('p2')	ppi = bi-1	for i=1, ..., 8	0
-2 ('m2')	ppi = ¬bi-1	for i=1, ..., 8	1

Note that b8 = b7 - sign bit of multiplicand; and pp8 the sign bit of partial products

Let us now detail the different decoder and selector schemes that produce the partial products. From table 2 and table 3, in order to select the correct bit of the multiplicand, we need four signals (ml, m2, pl and p2 meaning: minus one, minus two, plus one and plus two respectively) This way we obtain equations 1 and 2. for the partial products.

ppj = (pl bj) + (p2 bj-1) + (ml ¬bj) + (m2 ¬bj-1) Equation 1.

add1 = m1 + m2 Equation 2.

Figure 3 shows the corresponding implementation [Ara89], where signals ml, m2, pi and p2 are generated by the decoding cell.

Table 2 - Relationship Between Partial Product and Recoded Signed Bits

Multiplier Recoded Digit	Partial Products									add	Remarks
	pp8	pp7	pp6	pp5	pp4	pp3	pp2	pp1	pp0		
0 ('0')	0	0	0	0	0	0	0	0	0	0	all 0's
+1 ('p1')	b7	b7	b6	b5	b4	b3	b2	b1	b0	0	B
-1 ('m1')	¬b7	¬b7	¬b6	¬b5	¬b4	¬b3	¬b2	¬b1	¬b0	1	Invert B & add 1 to LSB
+2 ('p2')	b7	b6	b5	b4	b3	b2	b1	b0	b-1	0	Shift B one position left
-2 ('m2')	¬b7	¬b6	¬b5	¬b4	¬b3	¬b2	¬b1	¬b0	¬b-1	1	Shift B , Invert & add 1 to LSB

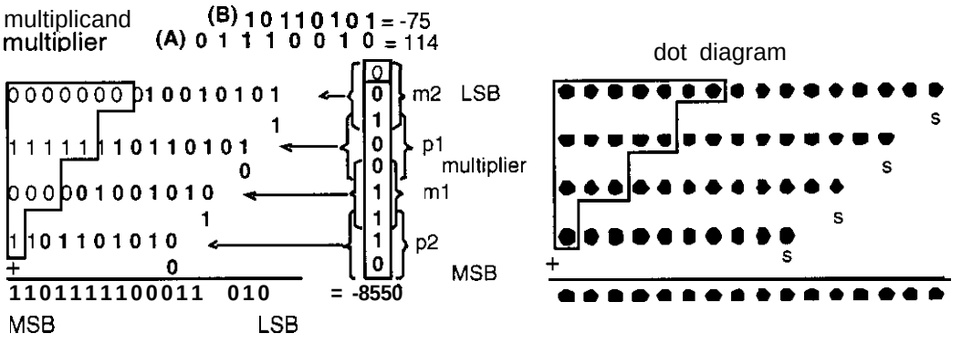

Figure 1 - A 8x8 Booth2 Signed Multiplication

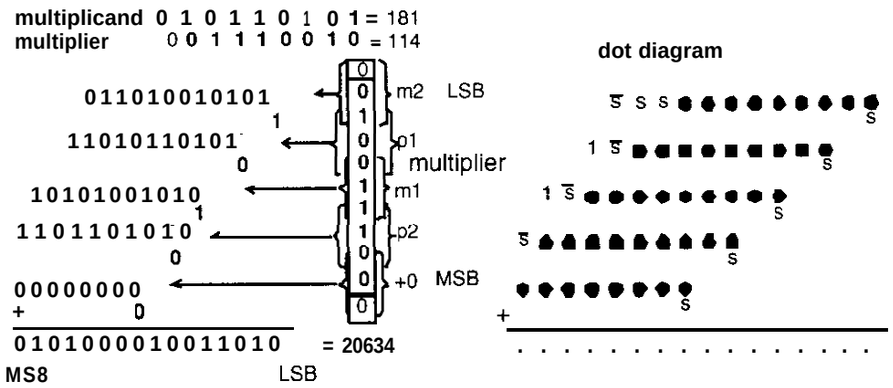

Figure 2 - A 8x8 Booth2 Non-Signed Multiplication

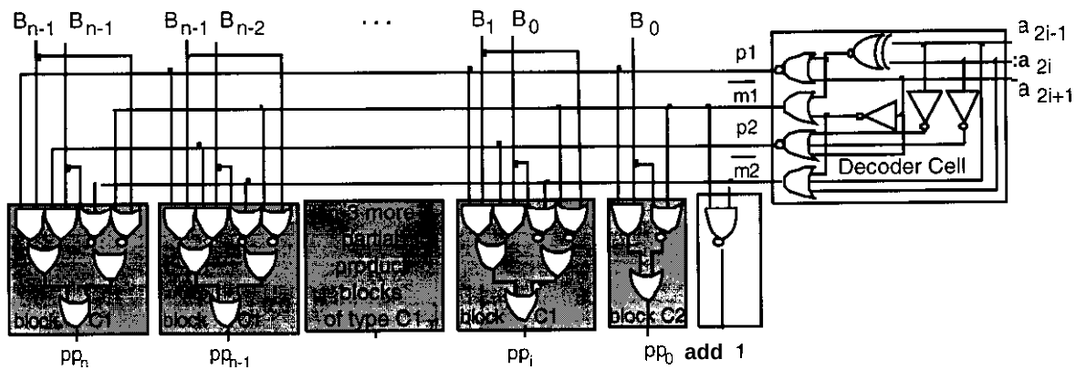

Figure 3 - First Implementation for the Decoder + Selector Sooth2 Circuit

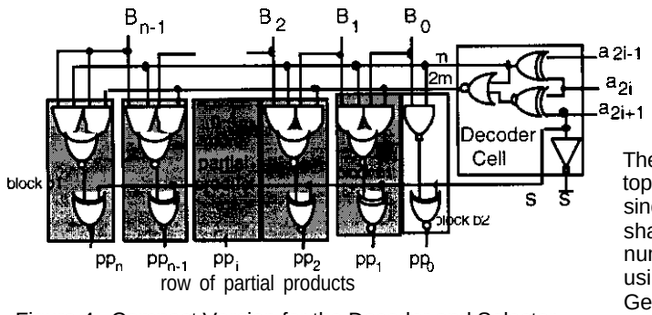

Figure 4 - Compact Version for the Decoder and Selector Booth2 Circuits

An alternative way of generating the partial products is represented in figure 4 [Wes94]. This circuit presents a more compact decoder + selector circuit implementation, but it introduces a slight additional delay on the partial product generation. m, 2m and s represent respectively: add the multiplicand, multiply by 2 the multiplicand and complement the obtained bits.

II.1 Booth Recoding with Carry-Save Topologies

From now on, we will concentrate on the signed multiplication scheme represented in figure 1. since this is the general notation adopted for implementing Booth multipliers. However, our solutions are trivially adapted to the non-signed multiplication. Note from figure 1.b that the partial products from one row to another are physically shifted two positions left due to the recoding process. Also, a sign extension is necessary to .Produce correct results (represented by the non-bolded surrounded digits - in figure 1). The dot diagram (figure 1) is mapped directly in a carry-save topology. However this direct mapping is hardware costly, since the sign extension provide a multiplier with a trapeze shape and the problem becomes more serious when the number of bits grows. This sign extension problem is solved using two techniques. The Signal Propagate and The Signal Generate reported in [Ara89]. These techniques provide hardware reduction and a multiplier implementation with appropriate rectangular shape. Summarizing, it consists on propagating the signal (most significant bit) of one row to the two most significant bits of the next row. In addition we need to eliminate the influence of the carry, generated by the adder cell that generates the signal, on the two most significant bits of the following row. Figure 5.a represents the complete carry-save implementation of a 8x8 Booth2 multiplier using the decoder and selector circuits of figure 3. Figure 5.b shows the same multiplier for the decoder and selector circuits of figure 4. The shaded squares represent HAs and the blank squares FAs. The cl blocks represent the selector circuits and c2 block the add 1 signal from equation 2. The last adder stage is a ripple-carry adder (RCA).

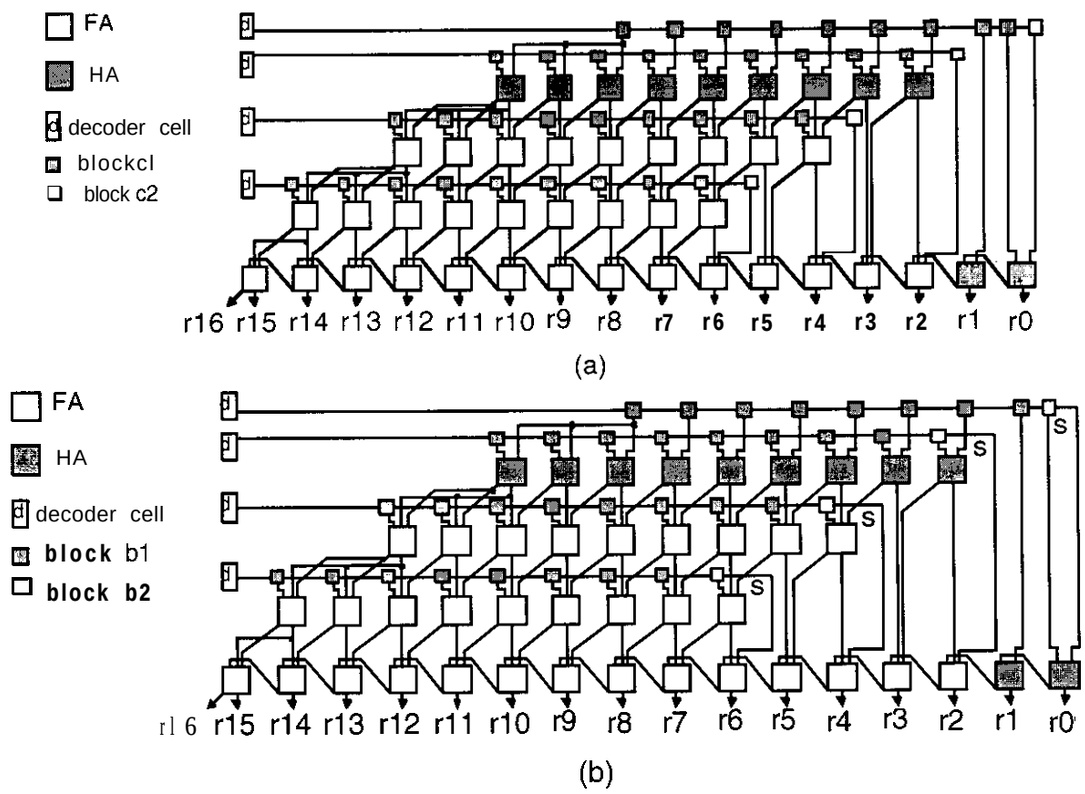

Figure 5 - Carry-Save Implementations for a 8x8 Booth2 Multiplier

III. Parity Prediction in Booth Multipliers

Practical multiplier designs, are implemented by a circuit generating the partial products and a network of full and half adders performing the summation of the partial products. Since the sum output of a full or half adder cell is equal to the modulo 2 sum of its inputs, this output gives the parity of the inputs of the cell. Thus, we can find trivially that the modulo sum of the multiplier outputs (output parity Pout), is equal to (parity of the product terms) XOR (parity of the carries of the full and half adder cells). That is Pout=PppXORPC. When operand recoding is not used, the partial product ppij is equal to $a_i \wedge a_j$. We find trivially that the modulo 2 sum of the terms $a_i \wedge a_j$ is equal to $(PA \wedge PB)$, and $Pout=(PA \wedge PB) \oplus P_C$. Where PA, PB are the parities of the input operands and PC is the parity of the internal carries. The Booth multipliers are also composed of a partial product generator and a network of adder cells. Thus, the relationship Pout = Ppp XOR PC still holds. However, the computating Ppp is more complex, due to the operand recoding.

We saw previously that the partial product generator in a Booth multiplier is composed of a set of decoder cells and a set of 'selector rows, each selector row being controlled by a decoder cell. Figure 6 presents the block diagram of these parts together with the adder-cell network (Sum of Partial Products).

In order to generate the parity of the Ppp of the partial products, we will first generate the parity of the partial products of each row. We have to elaborate the solution for the two approaches of partial product generation shown respectively in figures 3 and 4.

The modulo 2 sum of the partial products of a row i (i.e. the modulo 2 sum of the terms of equation 1 over j and of the sign extension), can be expressed as:

$$Pppi = \sum_{j=0}^{n-1} ppij = \sum_{j=0}^{n-1} (pli\ bj \vee p2i\ bj\text{-}1 \vee m1i \neg bj \vee m2i \neg bj\text{-}1)$$

$$\oplus \sum_{j=n}^{2n-1} (p1i\, bn\text{-}1 \vee p2i\, bn\text{-}1 \vee mli\, bn\text{-}1 \vee m2i \neg bn\text{-}1)$$

Since only one of the terms pli. p2i, mli, and m2i can be equal to 1 at a time, this sum can be expressed as:

$$p1i(\sum_{j=0}^{n-1} bj \oplus \sum_{j=n}^{2n-1} bn\text{-}1) \vee p2i(\sum_{j=0}^{n-1} bj\text{-}1 \oplus \sum_{j=n}^{2n-1} bn\text{-}1) \vee$$

$$m1i(\sum_{j=0}^{n-1} \neg bj \oplus \sum_{j=n}^{2n-1} \neg bn\text{-}1) \vee m2i(\sum_{j=0}^{n-1} \neg bj\text{-}1 \oplus \sum_{j=n}^{2n-1} \neg bn\text{-}1) \oplus$$

(mli v m2i).

Note that mli v m2i is also added since it gives the bit add1 (equation 2). For n by n multipliers we have 2n inversions in the 3rd and 4th terms, and they disappear from the modulo 2 sums. By taking also into account that $b_{-1} = 0$, we obtain:

$$Pppi = [p1i(PB \oplus \sum_{j=n}^{2n-1} bn\text{-}1) \vee p2i(PB \oplus \sum_{j=n}^{2n-2} bn\text{-}1) \vee mli$$

$$(PB \oplus \sum_{j=n}^{2n-1} bn\text{-}1) \vee m2i\,(PB \oplus \sum_{j=n}^{2n-2} bn\text{-}1)] \oplus (m1i \vee m2i)$$

$$= [(pli \vee m1i)(PB \oplus \sum_{j=n}^{2n-1} bn\text{-}1) \vee (p2i \vee m2i)(PB \oplus \sum_{j=n}^{2n-2} bn\text{-}1)] \oplus$$

(mli v m2i), where PB is the parity of the multiplicand.

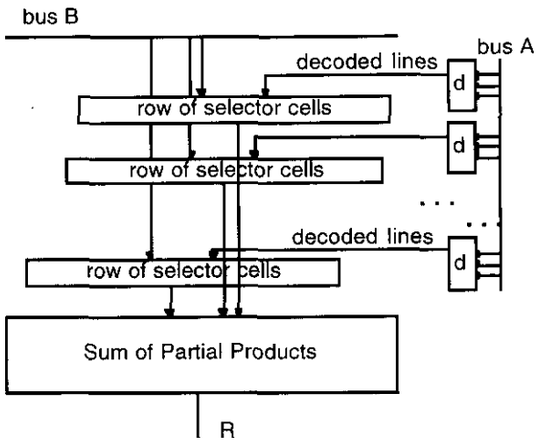

Figure 6 - Booth2 Multiplier - Block Diagram

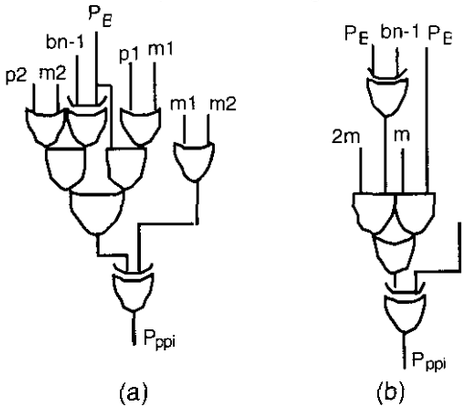

Figure 7 - Row Parity Prediction Circuit for Booth2

For n eve" we find: Pppi = [(pli v m1i)(PB) v (p2i v m2i)(PB⊕bn-1)] ⊕ (mli v m2i).

For n odd we find. Pppi = [(pli v m1i)(PB⊕bn-1) v (p2i v m2i)(PB)] ⊕ (ml i v m2i).

I" the following we consider the usual case where n is eve" and we obtain the circuit of figure 7.a.

Similarly, for the compact decoder circuit of figure 4 and n eve", we obtain the following parity prediction equation: Pppi = [miPB v 2mi(PB⊕bn-1)] ⊕ si (with si = a,,,,). This equations is implemented in figure 7b.

The parity Ppp of the partial products is computed as the modulo 2 sum of the partial product row parities Pppi = $\sum_{i=0}^{\lceil \frac{n+1}{2} \rceil - 1}$ Pppi. This is implemented by a network of XOR gates having as inputs the $\lceil (n+1)/2 \rceil - 1$ Pppi terms.

IV. Fault-Secure Implementation for Array Booth Multipliers

We saw that there are two different forms of implementing the decoder + selector circuit. The fault-secure solution proposed and analyzed in this section is the same for both implementations.

The fault-secure property will be shown by considering the following parts:

a) The decoder cells;
b) The parity prediction block;
c) The block Sum of Partial Products;
d) The selector cells;

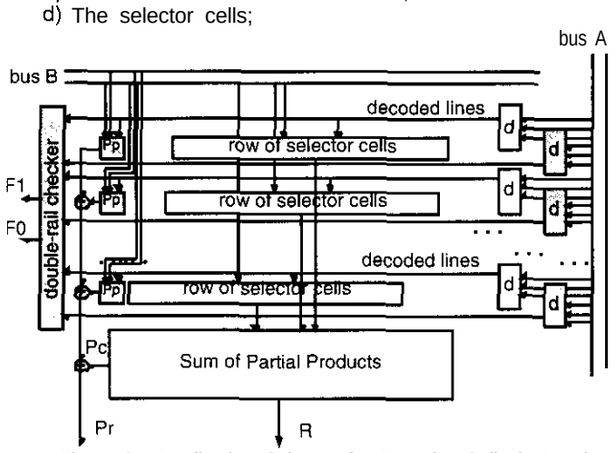

Figure 8 - Duplication Scheme for Decoder Cells in Booth Multipliers

a) A fault in a decoder cell provokes one or more erroneous decoded lines. These erroneous line(s) enter in all cells of a row of selection cells. This situation will result in multiple erroneous partial products. The multiple erroneous partial products create multiple erroneous inputs for the adder cell network. These multiple input errors will destroy the fault secure property. To cope with that we will use a duplication scheme to check the decoder cells. For each decoder cell its dual counterpart is also implemented, and the two parts are checked by a double-rail checker. This checker [Car68] produces two outputs FO, F1, indicating detection of eventual errors (see Figure 8).

b) The parity prediction part. Under a fault in this part, only the parity Pout is affected by the eventual errors, and the error is always detected.

c) Faults in the adder cells network. The analysis for this part is based on the results obtained in [NIC97]. These results are summarized below.

Single-Cell-Fan-out Networks: Let us call single-cell-fan-out network any cell-network in which each output of a cell enters exactly one input of exactly one cell. Many networks of full and half adders used in arithmetic operators verify this property.

As we have seen, the parity of an adder-cell network can be predicted using the equation Pout = Ppp ⊕ PC. The parity PC of the carries of the adder cell network is generated by using a network of XOR gates receiving as inputs the carry signals. In order to achieve the fault secure property, [NIC 97] shown that the full and half adder cells must verify some constraints. Two such cells was obtained. The one [figure 9) requires to use complete carry duplication and also implements the sum output by a separate circuit. The one of the duplicated carries is used for performing the addition function, while the second is used for parity prediction. The second cell (figure 10), is more compact since it was shown that the constraints can be relaxed to have the propagate signal (P= A XOR B) shared between the duplicated carries C. CP and the sum output S.

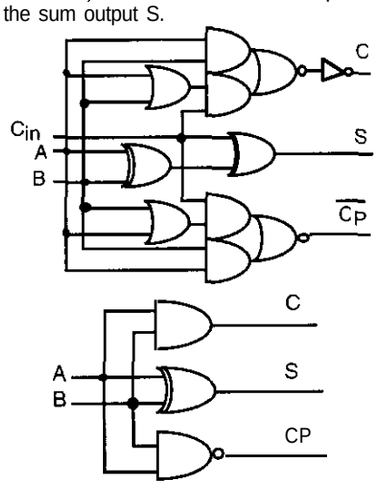

Figure 9 - Full- and half-adder cells for fault secure multiplier design.

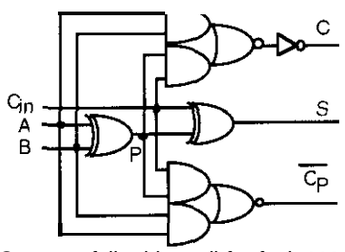

Figure 10 - Compact full-adder cell for fault secure multiplier design using logic sharing.

Theorem 1 and theorem 2 [NIC 97]: Two basic theorems [NIC 97] show that a single-cell fan-out network composed of full- and half-adders of figures 9 and 10. and checked by the parity prediction, meets the fault secure property

We have seen two possibilities for implementing the block *Sum of Partial Products* of figure 6.

C-I) Implementation using a network of adder cells arranged in a trapeze shape. .This implementation is a *direct* mapping of the partial products of figure 1 in an array of full and half adder cells arranged in a trapeze shape. The resulting circuit is a single-cell fan-out network and the fault secure property holds. However, as it was discussed before, this structure has a high hardware cost.

C-2) Implementation using a network of adder cells in a rectangular shape. This is the optimal solution in terms of hardware cost, since it eliminates the extra hardware required for generating the extension of the signal bit. For this case, the networks of figure 5 are analyzed. The analysis is similar for the implementation of figure 5.a and 5.b. Thus, in the following, we will only refer to figure 5.a. Unfortunately this circuit is not a single-cell fan-out

network. This property is not respected by the part concerning the signal extension. This part is amplified in figure 11. As we can observe there, there are two carries entering two cells each, two sum signals entering three cells each, and one sum signal entering two cells. To take care of this network we will consider our previous results obtained in [NIC 97] for multiple-cell fan-out networks. The following theorem holds when any output of a cell enters an odd number of inputs of other cells (odd-cell fan-out network).

Theorem 3 [NIC 97]: An odd-cell fan-out network composed of full- and half-adders of figures 9 and IO. and checked by the parity prediction, meets the fault secure property.

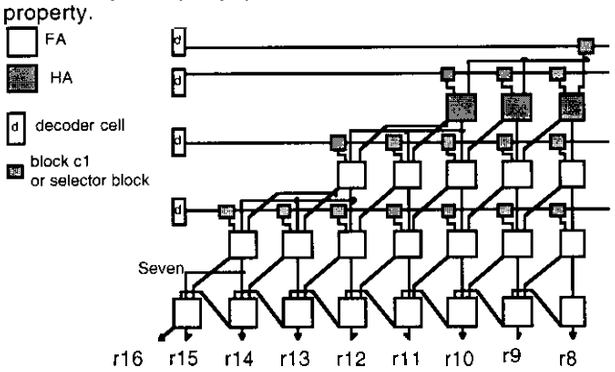

Figure 11 - Details of the Signal Extension Part of the Booth2 Multiplier of Figure 5

For networks with even-cell fan-outs the fault secure property is lost [NIC 97]. A solution proposed in [NIC 97] consists on duplicating and checking the signal with even-cell fan-out and all its predecessors. In figure 11, we have two carries and one sum signals with even-cell fan-out. Applying the above solutions to the sum signal entering the two left-most cells of the bottom row, will require to duplicate the whole array excepting the bottom row. This is because the outputs of all other rows are predecessors of this sum signal. This solution has an excessive hardware cost. For further analyzing this problem, let us first recall the concept of a sum path introduced in [NIC 97].

Sum-path: Consider a path starting from an input of an adder cell, finishing to an output of the network, and including only sum outputs of the network cells. Such a path will be called a sum-path.

In a single-cell fan-out there is exactly one sum-path starting from any input of any cell.

The following proposition and properties complete the results obtained in [NIC 97] and simplifies our search for a more compact solution. Prior to this proposition let us introduces the following concept.

Sum-path parity: A signal has an odd sum-path parity if it is connected to the outputs of the network through an odd number of sum paths, it has an even sum-path parity otherwise.

Proposition 1: In a network using the adder cells of figures 9 and 10. if each signal has an odd sum-path parity. then, the fault secure property is reached.

This proposition is proven similarly to theorem 3 in [NIC 971. Thus, the proof is omitted here.

When a signal has an odd- (resp. even-) cell fan-out, and all its successor sum signals have odd-cell fan-out, then, the sum-path parity of the signal remains odd (resp. even). However, if some of its successor sum signals have even-cell fan-out, then, we can show easily that the sum-path parity of the signal is determined by the following properties.

Property 1: An odd-cell fan-out signal which is predecessor of an odd number of sum signals with even-cell fan-out, has even sum-path parity.

Property 2: An odd-cell fan-out signal which is predecessor of an even number of sum signals with even-cell fan-out, has odd sum-path parity.

Property 3: An even-cell fan-out signal which is predecessor of an odd number of sum signals with even-cell fan-out, has odd sum-path parity.

Property 4: An even-cell fan-out signal which is predecessor of an even number of sum signals with even-cell fan-out, has even sum-path parity.

In figure 11. there is one sum signal with even-cell fan-out. This signal is labeled Seven in the figure. From Property 3. its predecessors with even-cell fan-out have odd sum-path parity. There are exactly two predecessors of Seven having even-cell fan-out. Thus, these signals verify the structure required from Proposition. These signals are the carry outputs of the left-most cells of the first and second rows in figure 11. From Property 1, all the other predecessors of Seven will have even sum-path parity and do not verify Proposition 1. For avoiding to duplicate and check all these signals, we will modify the network to meet the following two points:

1) Remove the even-cell fan-out from signal labeled Seven. This wav. this signal and all its predecessors with odd-cell fan-out will reach the requirements of Proposition 1.

2) The transformation of point 1- will modify from odd to even the sum-path parity of the two carry signals with even-cell fan-out. To avoid this new problem, we will modify the cell fan-out of these signals from even to odd by incrementing it.

3) Do not modify the cell fan-out of the remaining signals.

Figure 12 presents the modifications allowing to meet points i-:2- and 3-.

For meeting point 1-, the logic generating the signal Seven is duplicated. This signal is the sum output of a full adder cell. Thus the duplicated logic consists on a 3-input XOR gate.

This duplication will modify from odd to even the sum-path parity of the sum output of the cell feeding the duplicated logic. To avoid this problem and meet point 3- we duplicate the complete sum path of the multiplier passing from Seven. In the example of the 8 by 8 multiplier this requires to add two XOR gates. Two of the inputs of these gates come from duplicated selector cells (in bold) and its predecessor XOR gate. Thus. non of the existing sum signals of the multiplier enters any of the duplicated cells, and their sum-path parity is maintained. At the same time we use the carry signals with even sum-path parity as inputs to the duplicated logic. Thus their sum-path parity is modified from even to odd. and point 2- is met.

After these modifications, it is easy to check in figure 12 that the resulting circuit performing the Sum of Partial Products is an odd-cell fan-out network. This network uses

the adder cell of figure 9 or 10. It also uses some cells which are not adders (the three XOR gates). However, as shown in [NIC97] theorems 1, 2 and 3 hold if the cells of the network verify the following requirements:
Any error on a single input of a cell is propagated to the one output of the cell (say, the "sum" output). This error propagation implies that the "sum" output is computed as the XOR or XNOR function of the cell inputs. No particular property is required for the function of the other output.

Since the concerned cells are XOR gates, their output verifies the requirements of the "sum" output. Since there are no requirements for the other output of the cell, the cell requirements are met in the case of the XOR cell, where this output is missed.

From the above the fault secure property is reached for faults affecting the Sum of Partial Products network.

d) The selector cells. A fault in any selector circuit provokes an error on a single partial product. As we can check in figure 5, one partial product enters three cells and each other partial product enters a single cell. Thus partial products can be viewed as odd-cell fan-out signals in an odd-cell fan-out network. Thus the fault secure property holds also for faults in the selector cells.

The complete fault-secure solution for the optimal Booth2 multipliers (figure 5) are shown in figure 13. Figure 13.a shows the topology for the decoder + selector cell Of figure 3 and figure 13.b shows the topology for the decoder + selector cell of figure 4.

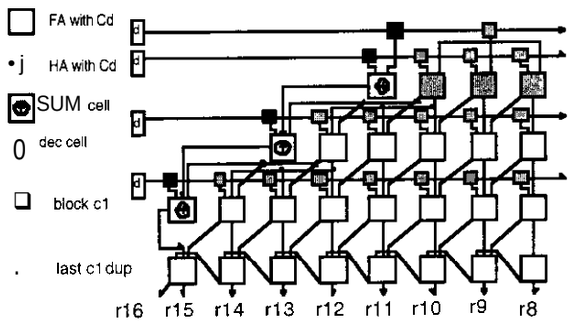

Figure 12 Solution for the Network Part due to the Signal Extension

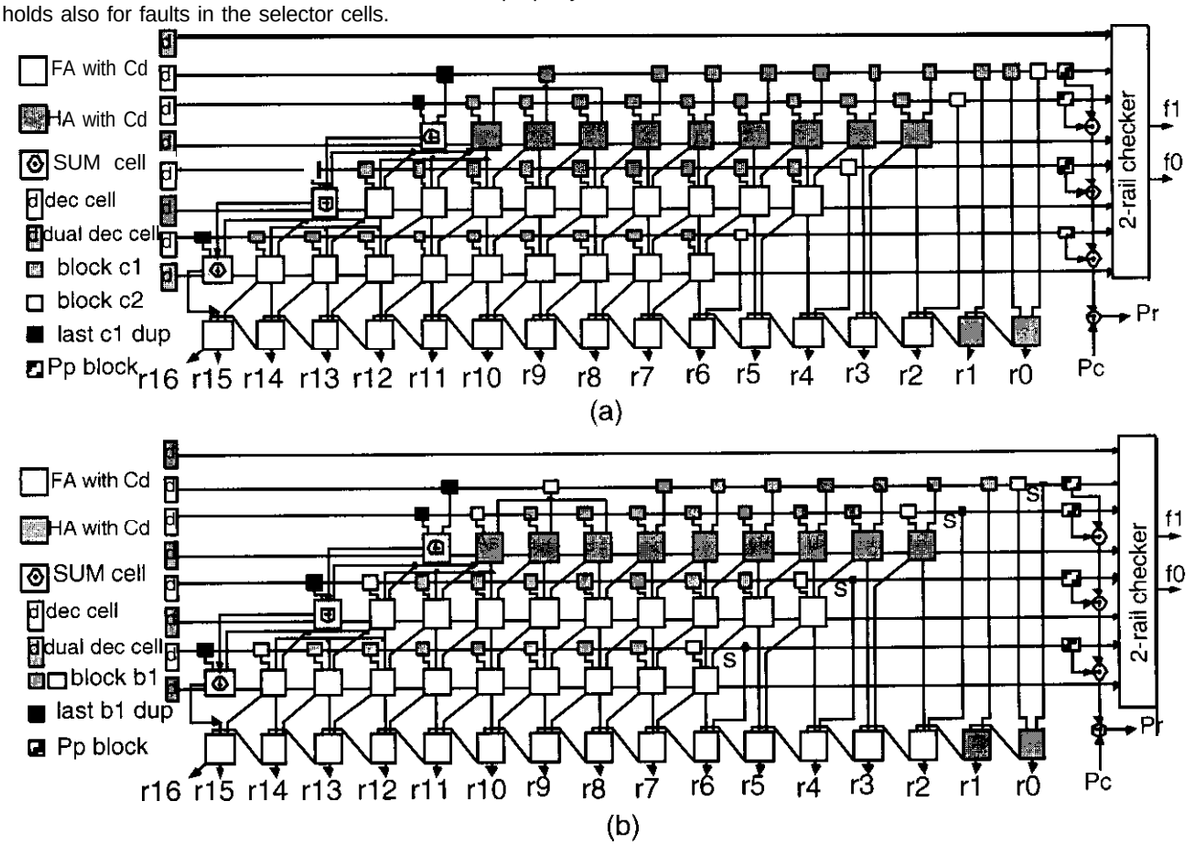

Figure 13 - Fault-Secure Solution for Carry-Save Topologies with Sooth2 Recoding

V. Cost Reduction

Next we are exploiting the 2-rail checker controlling the duplicated decoder cells, in order to remove the circuit predicting the parity for partial products. In section III we obtained the following equation predicting the parity of row i of partial products when we use the decoder of figure 3:
For n even: $Pppi = [(pli \vee m1i)(PB) \vee (p2i \vee m2i)(PB \oplus bn\text{-}1)] \oplus (m1i \vee m2i)$. Since only (pli v mli) or only (p2i v m2i) can

be equal to 1 at a time, we find Pppi = [(p1i v mli v p2i v m2i)PB ⊕ (p2i v m2i) bn-1] ⊕ (ml i v m2i). Again, since only one of the terms pli, mli, p2i, m2i can be equal to 1 at a time we can replace the OR function by the XOR function. Thus we find Pppi = [(pli ⊕ mli ⊕ p2i ⊕ m2i)PB ⊕ (p2i ⊕ m2i)bn-1] ⊕ (mli ⊕ m2i).

The modulo 2 sum of the terms Pppi over the set of rows i gives the parity Ppp of the partial products. That is, Ppp = $\sum_{i=0}^{n-1}$ Pppi = $\sum_{i=0}^{n-1}$ (pli ⊕ mli ⊕ p2i ⊕ m2i)PB ⊕ (p2i ⊕ m2i)bn-1⊕ (mli ⊕ m2i)

3 P p p = (P p l ⊕ Pm1 ⊕ Pp2 ⊕ Pm2)PB ⊕ (Pp2 ⊕ Pm2)bn-1⊕ (Pm1⊕ Pm2).

Where Ppl, Pml. Pp2, Pm2 are the parities of the terms p1i's, mli's, p2i's, m2i's. To compute these parities we can exploit the properties of a double-rail checker in order to use this checker for both check a set of double-rail signals [NIC 93] and generate their parity. The two outputs of the double-rail checker will indicate any discrepancy on the double-rail signals. At the same time one of the checker outputs provides the parity of these signals. To exploit the checker verifying the decoder cells (figure 8). we will implement it by using four double-rail checker trees. These trees check respectively the terms pli, mli. p2i and m2i. Thus, using one output of each of them we obtain the parities Ppl. Pml, Pp2 and Pm2. Three double-rail checker cells combine the outputs of the trees to generate the error indication for faults in the decoder cells. The signals Ppl, Pml, Pp2 and Pm2 are used to generate the parity Ppp of the partial products (figure 14). As we see there we only need to add two AND gates and two XOR gates for generating Ppp. This reduces considerably the cost since the cells of figure 7 and the parity tree combining the Pppi's are eliminated.

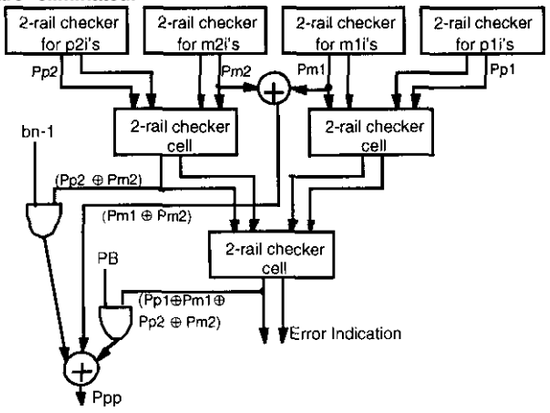

Figure 14 - Exploiting the double-rail checker for generating Ppp for the case of the decoder of figure 3.

Similarly for n odd we find: Ppp = (Ppl i ⊕ Pmli ⊕ Pp2i ⊕ Pm2i)PB ⊕ (Ppli ⊕ Pm1i)bn-1] ⊕ (Pmli ⊕ Pm2i). Thus we use a similar implementation for generating the parity of the partial products.

Similarly, for the compact decoder cell of figure 4 and for n even, from the equation Pppi = [miPB v 2mi(PB⊕bn-1)]⊕si, we obtain: Ppp = (Pm⊕P2m)PB⊕P2m bn-1)]⊕Ps. Where Pm, P2m and Ps are the parities of the terms mi, 2mi, and si. As in the previous case, we can use the outputs of the modules of the double-rail checker to obtain the parities Pm and P2m and Ps. We find that we need to add only two AND and one XOR gate to the double-rail checker in order to obtain the parity of the product terms. It results in a significant cost reduction.

IX. CONCLUSIONS

In this work we have analyzed the structure of Booth multipliers and derived the parity prediction equations and circuits. Then, on the basis of a theory proposed recently [NIC97], we have derived the fault secure implementation for these designs. Some extension of this previous theory and some specific modifications of the multiplier was necessary in order to cope with even-cell fan-out signals.

REFERENCES

[ARA89] M. Annaratone, "Digital CMOS Circuit Desing", Kluwer Academic Publishers. 1989

[AVI73] Avizienis A., 'Arithmetic Algorithms for Error-Coded Operands' IEEE TC, Vol. C-22, No. 6, pp.567-572, June 1973.

[BOO51] A.D. Booth, "A signed Binary Multiplication Technique", Quarterly Journal of Mechanics and Applied Mathematics. N°4, pp. 236-240, June 1951

[BUR94] B. Burgess, M. Alexander. Y.-W. Ho. S.P. Litch, and Cols., "The PowerPC™603 Microprocessor: A High Performance, Low Power, Superscalar RISC Microprocessor', Design Automation Conference DAC'94, pp. 330-306, 1994

[CAR68] W.C. Carter, P.R. Schneider, "Design of Dynamically Checked Computers", in Proc. IFIP Conf., Edinburgh. Scotland. Aug. 1968, pp.878-883

[DUA 97] Duarte R. O, Nicolaidis M., "A CAD Framework for Efficient Self-Checking Data Path Design", 3rd IEEE Intl. On-Line Testing Workshop, Aghia Pelaghia, Crete, Greece, July 1997.

[MAK96] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara and K. Mashiko, "An 8.8-ns 54 x 54-bit Multiplier with High Speed Redundant Binary Architecutre", IEEE Journal of Solid-State Circuits, vol. 31, n°6, June 1996

[McS61] O.L McSorley, "High Speed Arithmetic in Binary Computers", in Proc. of the IRE, pp. 67-91 January 1961

[MOR91] J. Mori, M. Nagamatsu, M. Imai, S. Tanaka, M. Noda, et al., "A 10-ns 54x54 Parallel Structured Full Array Multiplier with 0.5µm CMOS Technology', IEEE Journal of Solid-State Circuits, vol. SC-26 N°4, pp. 600-605, April 1991

[NIC93] Nicolaidis M. 'Efficient Implementation of Self-Checking Adders and ALUs'. Proc. 23th Fault Tolerant Computing Symposium, Toulouse France, June 1993.

[NIC 97] Nicolaidis M., Duarte R.O., Manich S., Figueras J., "Fault Secure Parity Prediction Arithmetic Operators". IEEE Design and Test of Computers April-June 1997.

[SAT91] T. Sato, N. Nakajima, T. Sukemura, G. Goto, "A Regularly Structured 54-Bit Modified Wallace-tree Multiplier". VLSI Design Conference, pp. 1.1.1-1.1.9, 1991

[SEL68] Sellers F.F. et all, 'Error Detecting Logic for Digital Computers', New-York: McGraw-Hill 1968.

[TWA95] H. Al-Twaijry and M. Flynn, "Performance/Area Tradeoffs in Booth Multipliers". Tech". Rep.: CSL-TR-95-684, November 1995

[WES94] N.H.E Weste, K. Eshraghian, Priciples of CMOS VLSI Design A Systems Perspective - scond edition, Addison-Wesley Publishing Co., 1994

[YU95] R.K. Yu and G.B Zyner, "167 MHz Radix-4 Floating Point Multiplier', in Proc. 14th Symposium on Computer Arithmetic, pp. 149-154, 1995