Int. J. Appl. Math. Comput. Sci., 2008, Vol. 18, No. 3, 319–328
DOI: 10.2478/v10006-008-0029-4

NEW SELF-CHECKING BOOTH MULTIPLIERS

MARC HUNGER [*], DANIEL MARIENFELD [**]

[*] Department of Electrical Engineering and Computer Science
University of Paderborn, 33098 Paderborn, Germany
e-mail: mhunger@date.upb.de

[**] Department of Computer Science
University of Potsdam, 14415 Potsdam, Germany
e-mail: dmarien@uni-potsdam.de

This work presents the first self-checking Booth-3 multiplier and a new self-checking Booth-2 multiplier using parity prediction. We propose a method which combines error-detection of Booth-3 (or Booth-2) decoder cells and parity prediction. Additionally, code disjointness is ensured by reusing logic for partial product generation. Parity prediction is applied to a carry-save-adder with the standard sign-bit extension. In this adder almost all cells have odd fanouts and faults are detected by the parity. Only one adder cell has an even fanout in the case of Booth-3 multiplication. Especially, for even-number Booth-2 multipliers parity prediction becomes efficient. Since that prediction slightly differs from previous work which describes CSA-folded adders, formulas to predict the parity are developed here. The proposed multipliers are compared experimentally with existing solutions. Only 102% of the area of Booth-2 without error detection is needed for the self-checking Booth-3 multiplier.

Keywords: Booth multiplier, self-checking, parity-prediction, carry-dependent adder, 1-out-of-5 code.

1. Introduction

Transient faults caused by electrical noise or external radiation are of growing importance and must be detected on-line. As (Shivakumar *et al.*, 2002) describes, they result in soft errors in output latches of a combinational circuit if:

1. an output depends on the faulty subcircuit with respect to the input (logical condition);
2. a pulse, resulting from faults, has a significant duration and amplitude (electrical condition);
3. a pulse, resulting from faults, arrives at the latches at the clock transition (latching window).

Since there are a lot of masking effects, transient faults usually result in single bit errors. These effects become smaller for faults in latches or faults in the logic near latches. Therefore circuits which detect single input faults, single stuck-at-faults and multiple output faults are of interest.

This paper presents a self-checking Booth-3 and -2 multiplier. Since data paths usually use the parity code and/or double rail code for error detection, we developed a parity checked multiplier with duplicated output.

In Booth multipliers the number system of one operand is to be changed by some simple decoding steps. Therefore, effort for additions can be reduced, but depending on the algorithm, hard multiples of one factor have to be generated. The proposed self-checking Booth-3 multiplier extends the output of each decoder cell to the 1-out-of-5 code, which is used in combination with the parity of multiplier X to detect faults in decoder cells and input faults. To generate the hard multiple $3 * Y$ and to check multiplicand Y by its parity, the sum-bit-duplicated look-ahead-adder (Ocheretnij *et al.*, 2001) is modified. The proposed Booth-2 multiplier uses an existing solution presented in (Marienfeld *et al.*, 2005) to check decoder cells.

In both multipliers, as a final Carry-Propagate-Adder (CPA), a sum-bit duplicated Carry-Ripple-Adder (CRA) conforming to (Marienfeld *et al.*, 2004) is used together with parity prediction of a Carry-Save-Adder (CSA) consisting of carry-depended adder cells and realizing the standard sign extension.

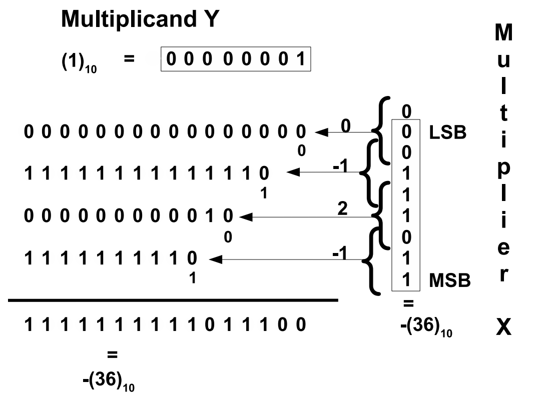

Fig. 1. Example of Booth-2 8-bit signed multiplication.

The following sections briefly describe basics of the Booth multiplier and self-checking adder networks. Section 4 proposes a new multiplier and Section 5 gives experimental results. Basics on computer arithmetics are presented in (Parhami, 2001), and (Lala, 2001) describes self-checking digital design.

2. Booth multiplier

Booth multipliers save costs (time and area) for adding partial products. They are reported, e.g., in (Booth, 1951; Al-Twaijry and Flynn, 1995). Figure 1 shows an example of signed Booth-2 multiplication. The multiplier $X = X_{n-1} \dots X_0$ is transformed from the two's complement to a radix-4 Booth code. The multiplicand $Y = Y_{n-1} \dots Y_0$ is multiplied with these digits to generate partial products of the form $Y * [\pm 0, \pm 1, \pm 2]$. These partial products are in two's complement representation and the most significant bit (MSB) serves as the sign-bit. The numbers are weighted and added in a combinatorial or sequential fashion using an adder network to form the product $P = P_{2*n-1} \dots P_0$. With the higher radix the number of additions is reduced and the redundant Booth code reduces costs for generating partial products in a higher radix system. As Fig. 1 shows, the number of summands is halved in contrast to the classical binary multiplication. Additionally, each partial product can be generated via simple shifts and bit inversion.

Figures 2 and 3 show the logical architecture of combinatorial Booth-2 and -3 multipliers. These circuits include a Partial-Product-Generator (PPG), a Carry-Save-Adder (CSA) and a final Carry-Propagate-Adder (CPA). The PPG decodes operand X into the Booth code with a higher radix. In the case of Booth-2, the radix is 4 and each decoded digit is in the set $[-2 : 2]$. Booth-3 uses a radix of 8 with digits $[-4 : 4]$. Each digit is decoded into selection signals $Seli$, which represent the multiple to select and the signal N, which is needed for inversions. Thus decoding partial products can be efficiently selected (Figs. 4 and 5 describe decoding and selection logic for Booth-2 and -3; these circuits result from Tables 2 and 3).

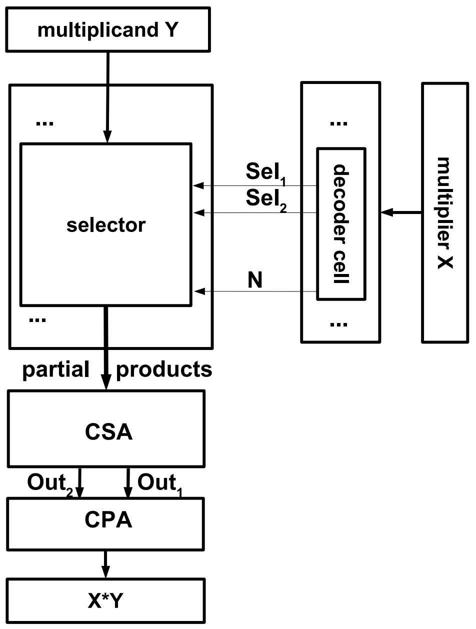

Fig. 2. Booth-2 multiplier.

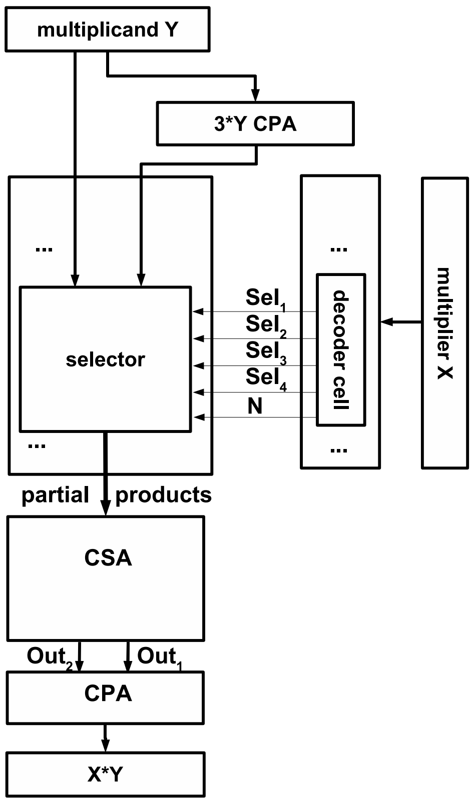

Fig. 3. Booth-3 multiplier.

The hard multiple $3 * Y$ needed in Booth-3 multi-

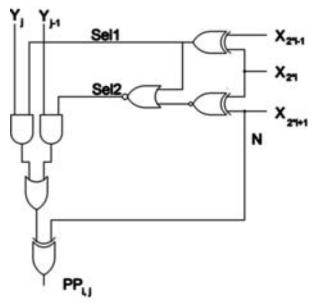

Fig. 4. Booth-2 selection circuit and decoder.

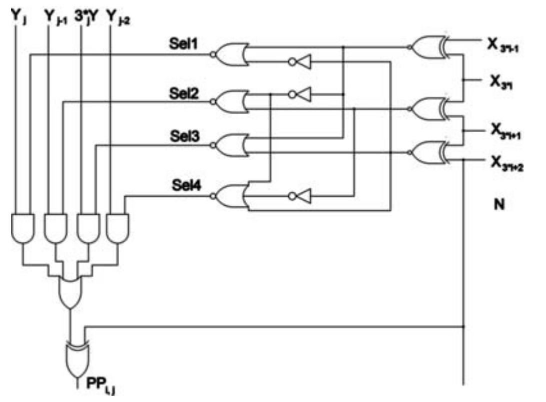

Fig. 5. Booth-3 selection circuit and decoder.

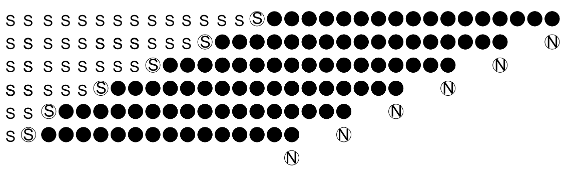

Fig. 6. 16-bit Booth-3 with full sign-extension.

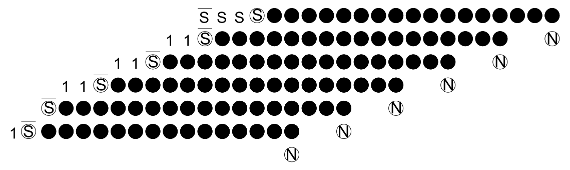

Fig. 7. 16-bit Booth-3 with CSA using correction terms.

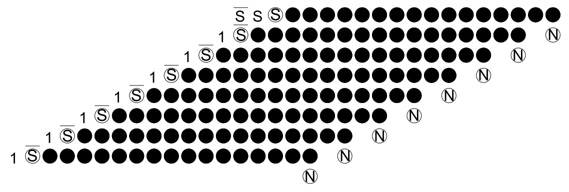

Fig. 8. 16-bit Booth-2 with CSA using correction terms.

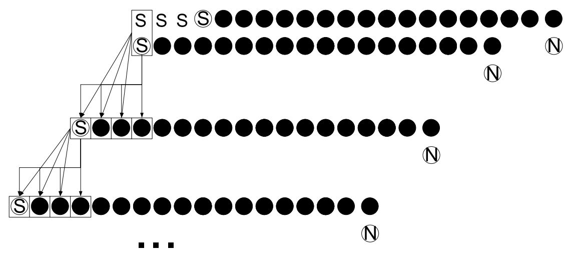

Fig. 9. 16-bit CSA-folded Booth-3.

pliers is generated using a CPA and all other multiples are generated by simple logical operations. The CSA reduces all partial products to two summands, which are added by the final CPA into the product P. There exist various structures to perform these kinds of additions based on full or half adders. The CSA can be realized as a tree or an array (Al-Twaijry and Flynn, 1995). The methods differ in their regularity, delay, area usage, wiring effort or power consumption. The CPA (Parhami, 2001) can be realized as a simple chain of full adders (carry-ripple-adder). It is possible to fasten carry-propagation by additional look-ahead or skip units. Alternatively, carries can be generated speculatively. The correct result can be obtained using a multiplexer. In this work, the CPA is realized as a carry-ripple-adder and a linear array serves as the CSA. To generate the hard multiple $3 * Y$, we choose a fast carry-look-ahead adder. This fastens the component with a small drawback in the area usage. But the described methods are independent of that choice.

Since in Booth multipliers the partial products are signed (including both integer and natural multiplications), they need to be extended to a $2 * n$ bit length. In two's complement representation an extension is performed by a sign-bit extension as shown in the DOT-scheme in Fig. 6. Here dots represent the bits generated via selector cells. The resulting DOT-scheme is in trapezoidal form and can be optimized by two known solutions reported in (Sparmann and Reddy, 1994). The standard sign-extension, or a "CSA with correction terms", applies the equation

$$\begin{aligned} S\,S\,S\,S \bullet\bullet\bullet\bullet &= 1\,1\,1\,\overline{S}\bullet\bullet\bullet\bullet \\ &\quad + 0\,0\,0\,1\,0\,0\,0\,0 \end{aligned} \tag{1}$$

(or a similar one) to each partial product. The resulting constants are added and Fig. 7 serves as an invariant for Booth-3. The resulting Booth-2 DOT-scheme is shown in Fig. 8. Special adder cells add remaining ones with the corresponding bits of partial products. The number of DOTS and therefore the number of full adders are reduced.

The second solution, described in (Nicolaidis and Duarte, 1998) and applied to Booth-3 in Fig. 9, consists of adder cells with multiple fanouts (CSA-folding). Here we use the fact that all half adders, adding the sign-bits of

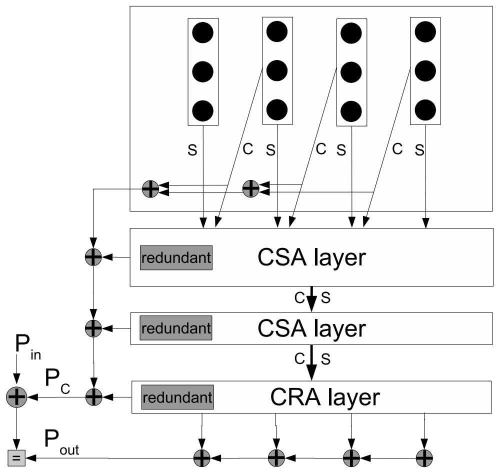

Fig. 10. Self-checking CSA with a final CRA.

the first two partial products, generate the same outputs. Therefore only one adder with multiple fanouts is needed. The same fact is true for adders in further rows. Thus in these rows redundant adder cells are saved, too. As in the first solution, the cost reduction is of significance. As described in Section 4.1, the parity of the partial products for standard sign-extension is different in comparison with a folded CSA.

3. Self-checking adders

To detect errors on-line, some kind of redundancy is needed. This includes time and information redundancy. In this work, with parity and the double rail code, information redundancy is used. Inputs I and outputs O for circuits are extended to code words, which are checked to detect errors. Codes for arithmetic circuits include, e.g., parity (the number of ones in I modulo 2), the Berger code (the number of zeros in I), as reported in (Lo *et al.*, 1993), Bose-Lin Codes, described in (Gorshe and Bose, 1996), or the residue code reported in (Sparmann and Reddy, 1994).

Figure 10 shows a parity checked CSA with a final CRA. The input code includes the operands and the parity of all summands (or partial products in the multiplier). The output code includes the sum and both the generated parity P_{Out} and the predicted parity given by

$$P_{\mathrm{Out}} = P_{\mathrm{In}} \oplus P_C, \tag{2}$$

where P_C is the parity of all carries.

For an on-line error detection, the following properties should be achieved, as described in (Goessel and Graf, 1993):

- **code-disjointness**: A circuit is called code-disjoint if each non code word is mapped to a non-output code word.

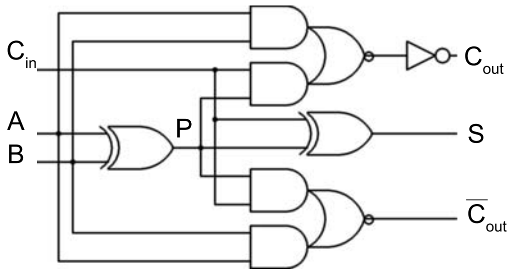

Fig. 11. Carry-duplicated adder.

- **fault-security**: A circuit is called fault-secure if for all faults (in the fault model) there is no input code word that causes the faulty circuit to output an incorrect code word.
- **self-testing**: A circuit is called self-testing if for all faults (in the fault model) there is at least one input code word such that the corresponding output is not a code word.
- **self-checking**: A circuit is called totally self checking if it is self-testing and fault-secure.

Code-disjointness ensures that faults in input latches are detectable. The fault-secure property ensures that the circuit under a fault either generates correct outputs or detects the fault. In self-testing circuits each fault can be tested by applying an input vector. To achieve this property, redundancies in circuits should be carefully analyzed.

For arbitrary adder cells, the adder described in Fig. 10 is not self-checking. Not detected are stuck-at-faults in adder cells, which affects only the carry-output. That faulty signal affects the generated parity via a sum output. The predicted parity is affected through P_C. Therefore the following redundant adder cells are needed in a self-checking adder:

1. A **carry-duplicated adder**, presented in (Nicolaidis *et al.*, 1997), duplicates the carries (Fig. 11). The propagate signal P can be shared by the sum-output and both carries. One of the carries is needed for carry-propagation and the other generates P_C and therefore predicts P_{Out}. That structure ensures that always odd or no outputs of the cell are erroneous.

2. A **carry-dependent adder**, proposed in (Hsiao and Sellers, 1963), forces a faulty carry to affect the corresponding sum-output. Here the sum-bit is realized as $S = f \oplus C$ (with f having a suitable truth table and a structure to meet the self-checking property). If f and C share no logic, then each fault affects either the signal f or C, or none of them. If C and f share an OR-gate 1, as shown in Fig. 12, the same property is saved. Also the special half adder, shown in Fig. 13, and the trivial cell adding $A + 1$ (here f is logical

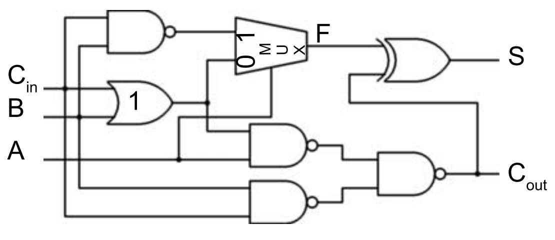

Fig. 12. Carry-dependent adder.

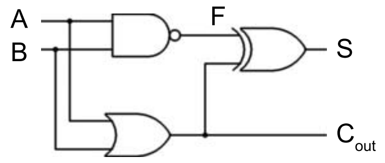

Fig. 13. Carry-dependent half adder: $A + B + 1$.

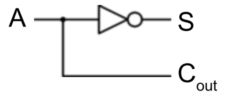

Fig. 14. Self-checking $A + 1$ cell.

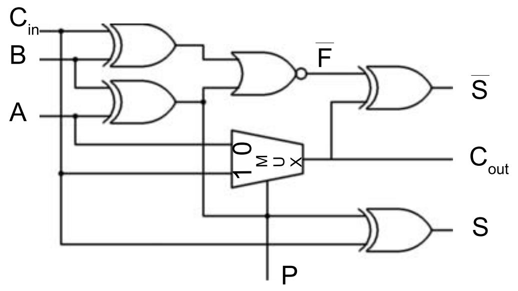

Fig. 15. Sum-bit duplicated carry-dependent adder.

one) preserve that property. So either no output or S or C together with S are faulty.

In this work a carry-dependent adder is used for the CSA since it gave better results in delay and area overhead.

Figure 15 shows a sum-bit duplicated carry-dependent adder cell used in the CPA given in Fig. 16. Note that the logic near latches is more sensitive to soft errors. Therefore that logic is partially duplicated and one of the sums is realized as carry-dependent. Since the carries are checked, a look-ahead adder without duplicated look-ahead units can be used. The parity of the propagated signals P is checked by the input parity. As shown in (Marienfeld *et al.*, 2004), a parity-checked carry-save-adder can be combined efficiently with a final sum-bit duplicated carry-dependent CPA to increase the error detection capability of a multiplier.

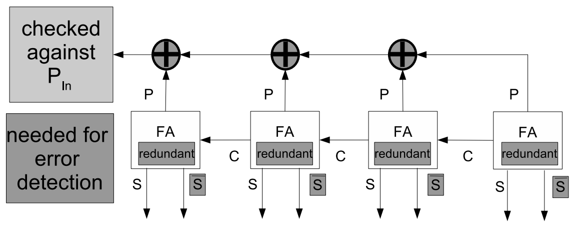

Fig. 16. Sum-bit duplicated carry ripple adder.

4. Proposed Booth multiplier

This section describes the proposed self-checking Booth-2 and Booth-3 multipliers. Caused by the sign-extension of Y and the decoding of X, parity prediction is not code-disjoint and so both inputs must be checked locally. Faults in decoder cells can generate multiple erroneous bits of one partial product. Faults in the $3 * Y$-CPA can generate multiple erroneous bits of multiple partial products. So both components need to be checked locally, too.

All faults in selector cells with odd fanout are detected by parity prediction. As Fig. 7 shows, one selector (generating the MSB of the first partial product in the Booth-3 multiplier) has an even fan-out. To meet the self-checking property, this selector is replaced by two selectors with odd fan-outs.

In both multipliers a linear array CSA with standard sign-extension using carry-dependent adder cells is used. The output-parity is checked by parity prediction. The final CPA is realized as a sum-bit-duplicated code-disjoint ripple adder. Faster methods for addition are possible.

Each multiplier needs to predict the parity of partial products. That prediction is combined with local decoder checks.

4.1. Self-checking Booth-2 multiplier. Figure 17 shows the architecture of the proposed Booth-2 multiplier. This multiplier is almost the same as that presented in (Marienfeld *et al.*, 2005), but a different CSA, and therefore different parity prediction, are applied. Operand Y is checked via a parity tree. The decoder is partially duplicated and checked by parity trees. These trees output needed signals to predict the output parity of the adder network.

The following describes the code-disjoint and self-checking decoder (Fig. 18):

1. The parity P_{Sel1} of all $Sel1_i$ is checked by the parity and the MSB of X.
2. The signals $Sel2$ of each decoder are duplicated and their parities P_{Sel2} and P_{Sel2D} are compared.

With these preparations, odd input faults and single stuck-at-faults are detected. At the same time, with P_{Sel1} and P_{Sel2}, signals to predict the partial products parity are computed. Additionally, P_Y is checked by a parity tree.

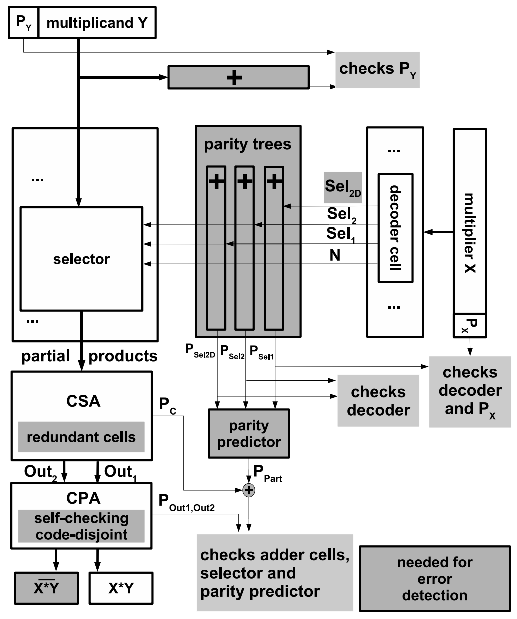

Fig. 17. Proposed Booth-2 multiplier.

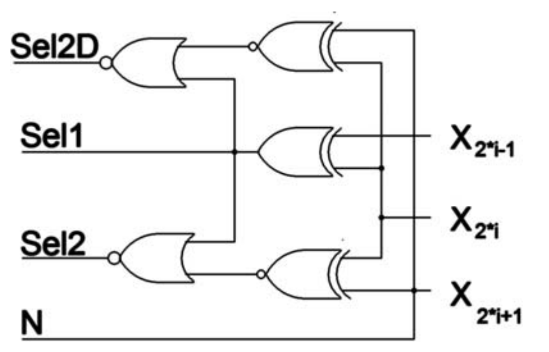

Fig. 18. Self-checking Booth-2 decoder cell.

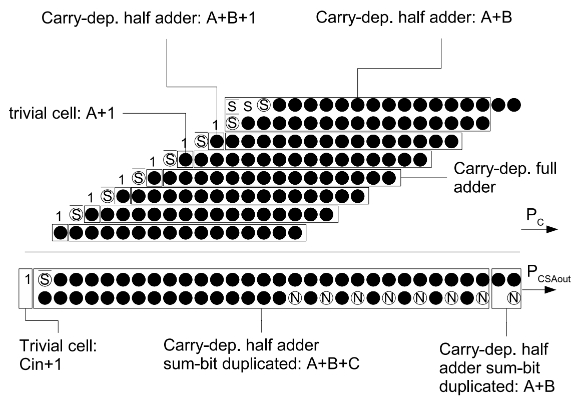
Carry-dep. half adder: A+B+1
Carry-dep. half adder: A+B
trivial cell: A+1
Carry-dep. full adder
P_C
P_CSAout
Trivial cell: Cin+1
Carry-dep. half adder sum-bit duplicated: A+B+C
Carry-dep. half adder sum-bit duplicated: A+B

Fig. 19. 16-bit self-checking Booth-2 CSA using correction terms.

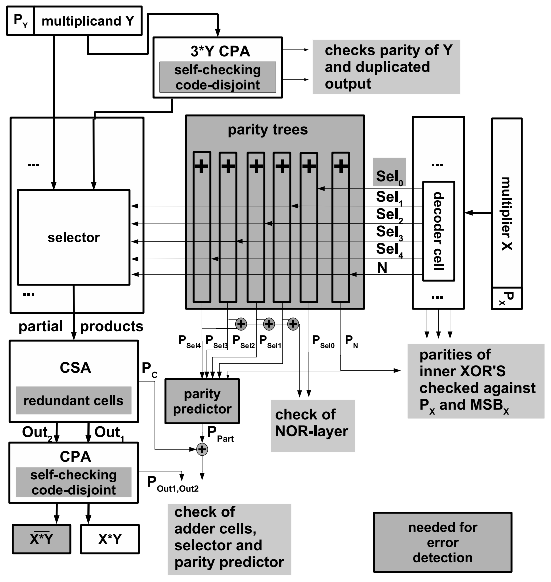
P_Y multiplicand Y
3*Y CPA
self-checking code-disjoint
checks parity of Y and duplicated output
parity trees
Sel_0
Sel_1
Sel_2
Sel_3
Sel_4
N
decoder cell
multiplier X
P_X
selector
partial products
P_Sel4 P_Sel3 P_Sel2 P_Sel1 P_Sel0 P_N
parities of inner XOR'S checked against P_X and MSB_X
CSA
redundant cells
P_C
parity predictor
check of NOR-layer
P_Part
Out_2 Out_1
CPA
self-checking code-disjoint
P_Out1,Out2
check of adder cells, selector and parity predictor
needed for error detection
X*Y
X*Y

Fig. 20. Self-checking Booth-3 multiplier.

In contrast to (Nicolaidis and Duarte, 1998), duplicating the decoder (and using a two-rail-checker) is avoided and costs are saved.

The CSA realizes the standard sign-extension instead of CSA-folding, as performed in (Marienfeld *et al.*, 2005; Nicolaidis and Duarte, 1998). The adder network is described in Fig. 19 and includes adder cells described in Section 3. For the standard sign-extended CSA, there are no cells with even fan-out, including adder and selector cells. Since in a CSA-folded Booth-2 multiplier some cells have even fan-outs, in (Nicolaidis and Duarte, 1998) the sum-circuit, including selector cells, of all most significant adder cells was duplicated to avoid even "sum-path parities". So the proposed method saves a parity tree and a set of selector cells.

As Fig. 19 shows, in case of "correction terms" the parity of the i-th partial product $(i > 0)$ is given by (3). The first term computes the parity of the lower bits (given by the selector cells and using the fact that both $Sel1_i$ and $Sel2_i$ are never equal to 1). The second term computes the parity of the sign-bit, which is inverted. The term N_i represents the signal added to the partial product in the least significant position and the last term 1 is added in the most significant position. This equation simplifies to

$$\begin{aligned} P_{\mathrm{Part}_{i>0}} &= \sum_{j=0}^{n-1}(Sel1_i * Y_j \oplus Sel2_i * Y_{j-1} \oplus N_i) \\ &\quad \oplus (Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus N_i \oplus 1) \\ &\quad \oplus N_i \oplus 1 \\ &= (Sel1_i \oplus Sel2_i) * P_Y \oplus Sel1_i * Y_{n-1}. \end{aligned} \tag{3}$$

The parity of the first partial product is

$$\begin{aligned}&P_{\mathrm{Part}_{i=0}}\\&=\sum_{j=0}^{n-1}(Sel1_i * Y_j \oplus Sel2_i * Y_{j-1} \oplus N_i)\\&\quad\oplus (Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus N_i)\\&\quad\oplus (Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus N_i)\\&\quad\oplus (Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus N_i \oplus 1)\\&\quad\oplus N_i\\&=(Sel1_i \oplus Sel2_i) * P_Y \oplus Sel1_i * Y_{n-1} \oplus 1.\end{aligned} \tag{4}$$

Here the last term 1 is saved. The last equality results from the fact that, since the sign-bit is tripled, it is added once to the parity (as in all other partial products).

The partial product parity is given by $\sum_i P_{\mathrm{Part}_i}$ as

$$\begin{aligned}&P_{\mathrm{Part}}\\&=\sum_{i=0}^{n/2-1}((Sel1_i \oplus Sel2_i) * P_Y \oplus Sel1_i * Y_{n-1})\\&\quad\oplus 1\\&=(P_{Sel1} \oplus P_{Sel2}) * P_Y \oplus P_{Sel1} * Y_{n-1} \oplus 1.\end{aligned} \tag{5}$$

This parity is independent of P_N. As shown in (Nicolaidis and Duarte, 1998), parity prediction in a CSA-folded Booth-2 multiplier for even length operands is performed by

$$\begin{aligned}&P_{\mathrm{Part\ CSA-folded}}\\&\qquad=(P_{Sel1} \oplus P_{Sel2})P_Y \oplus Y_{n-1}P_{Sel2} \oplus P_N.\end{aligned} \tag{6}$$

Since this equation depends on P_N, the proposed method does not requice the generation (by a parity tree or a two-rail-checker) of that signal.

Since a "CSA with correction terms" differs only in the cells given in Figs. 13 and 14, from a folded CSA the self-checking properties are preserved.

4.2. Self-checking Booth-3 multiplier.

Figure 20 shows the proposed Booth-3 multiplier. Since Booth-3 decoders are more complex and need more area, (partially) duplicating becomes more expensive. Additionally, the parity of X is not directly computable by decoder outputs.

The following method, presented in (Hunger, 2006), makes the Booth-3 decoder code-disjoint and self-checking.

1. Decoder outputs are extended by $Sel0$ to the 1-out-of-5 code (Fig. 21 and Tables 2 and 3). To combine code checking and parity prediction, for each $Seli$ an own parity tree forms P_{Seli}. These parities are checked by the equation

$$\begin{aligned}P_{Sel0} \overset{\neq}{=} P_{Sel1} &\oplus P_{Sel2}\\&\oplus P_{Sel3} \oplus P_{Sel4},\end{aligned} \tag{7}$$

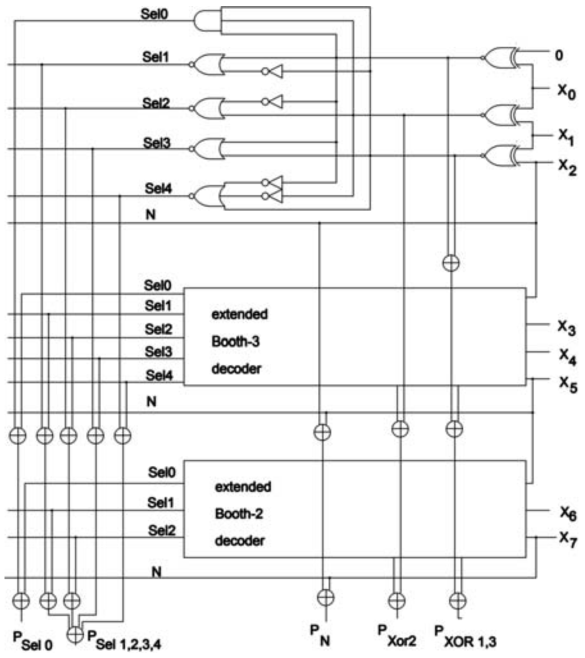

Fig. 21. Self-checking Booth-3 decoder.

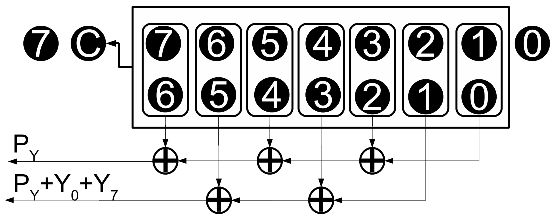

Fig. 22. Half-sums of 8-bit $3 * Y$-CPA.

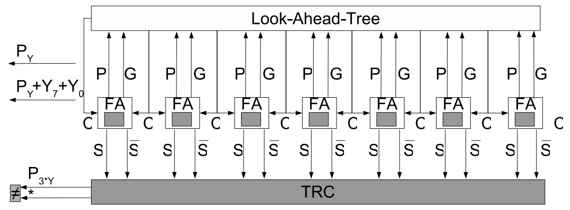

Fig. 23. Self-checking $3 * Y$-CLA.

where the equal sign is used if the number of decoder cells is even and the unequal sign is used otherwise.

2. The interior XOR-layer is checked by the parity of X. For a 8-bit decoder we use

$$P_X = P_{XOR2} \oplus P_N \oplus X_{n-1}, \tag{8}$$
$$P_X = P_{XOR1,3} \oplus P_N. \tag{9}$$

The following analyses the self-testing property of the decoder. The interior XOR-layer and the trees to compute P_{XOR2} and $P_{XOR1,3}$ can be tested, since all inputs of X can be applied. Also the NOR-layer and the parity of

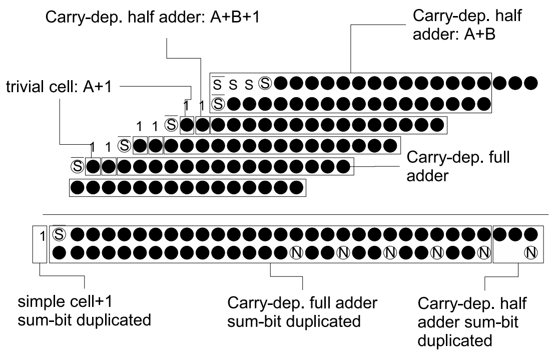
Carry-dep. half adder: A+B+1
Carry-dep. half adder: A+B
trivial cell: A+1
Carry-dep. full adder
simple cell+1 sum-bit duplicated
Carry-dep. full adder sum-bit duplicated
Carry-dep. half adder sum-bit duplicated

Fig. 24. 16-bit self-checking Booth-3 CSA using correction terms.

Table 1. Area overhead in % of the multipier.

algorithm | bits	8	16	32	64
Booth-2	100	100	100	100
Booth-3	108	101	91	88
Booth-2 acc. to Sec. 4	134	128	121	121
Booth-3 acc. to Sec. 4	156	132	109	102
Booth-2 acc. to (Nicolaidis and Duarte, 1998) ext. by a parity tree	184	165	151	142

the selection signals can be tested. By applying a number X mapped, e.g., to $(0)(+1)(0)$, and observing the parities P_{Sel0} and $P_{Sel\ 1,2,3,4}$, the NOR-layer of the second decoder cell can be tested. Under a stuck-at-0 fault in the NOR-gate computing $Sel1_1$, $P_{Sel\ 1,2,3,4}$ becomes zero. P_{Sel0} is not affected by that fault and is zero. So both signals are equal and the fault is detected.

Because of the sign-extension, the selector cell which forms the most significant bit of the first partial product needs to be duplicated.

To form the multiple $3 * Y$ and to check P_Y, the sum-bit duplicated code-disjoint look-ahead adder of (Ocheretnij *et al.*, 2001) is modified (Figs. 22 and 23). In this application, the odd and even half-sums are separately checked by the parity of Y. Therefore the whole multiplier becomes code-disjoint with respect to the parity code of Y. Although the inputs of that adder are restricted to the set of tuples of the form $(Y, 2 * Y)$, that adder is self-testing, including the two-rail checker which checks both sums. All possible inputs for adder cells can occur, since it can be chosen whether a carry is generated for cell i (only one vector of the second full adder is missing). The look-ahead tree consumes propagate P and generate signals G of adder cells and generates carries for these cells. For a look-ahead adder with arbitrary inputs, not all possible inputs into the look-ahead tree can occur. With the given restricted addition some additional inputs are missing. These are vectors in which one adder cell sets a generate signal and in one of its neighbours P and G are equal to zero. But using cascading standard look-ahead units of bit length 4, all faults are testable. For each bottom two-rail checker cell all needed inputs $\{(10,10), (01,01), 10,01), (01,10)\}$ can be generated to test that cell. Since each two-rail checker outputs the parity of its input (the output of the adder cells), the further two-rail cells can be tested, too.

Table 2. Booth-2-decoder.

X	N	$Sel_{1,2}$	$Sel0$
000	0	00	1
001	0	10	0
010	0	10	0
011	0	01	0
100	1	01	0
101	1	10	0
110	1	10	0
111	1	00	1

The "CSA with correction terms" is given in Fig. 24 and the parity of the partial product can be derived based on the following:

$$\begin{aligned}
&P_{\mathrm{Part}_{0<i<L}}\\
&= \sum_{j=0}^{n-1}\{Sel1_i * Y_j \oplus Sel2_i * Y_{j-1} \oplus Sel4_i * Y_{j-2}\\
&\quad \oplus Sel3_i * (3Y)_j \oplus N_i\}\\
&\quad \oplus \{Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus Sel4_i * Y_{n-2}\\
&\quad \oplus Sel3_i * (3Y)_n \oplus N_i\}\\
&\quad \oplus \{Sel1_i * Y_{n-1} \oplus Sel2_i * Y_{n-1} \oplus Sel4_i * Y_{n-1}\\
&\quad \oplus Sel3_i * (3Y)_{n+1} \oplus N_i \oplus 1\} \oplus N_i \qquad (10)\\
&= \sum_{j=0}^{n-1}\{Sel1_i * Y_j \oplus Sel2_i * Y_{j-1} \oplus Sel4_i * Y_{j-2}\\
&\quad \oplus Sel3_i * (3Y)_j \oplus N_i\}\\
&\quad \oplus Sel4_i * (Y_{n-2} \oplus Y_{n-1})\\
&\quad \oplus Sel3_i * ((3Y)_n \oplus (3Y)_{n+1}) \oplus N_i \oplus 1\\
&= Sel1_i * P_Y \oplus Sel2_i * P_Y \oplus Sel4_i * P_Y\\
&\quad \oplus Sel3_i * P_{3Y} \oplus Sel2_i * Y_{n-1} \oplus N_i \oplus 1\\
&= (Sel1_i \oplus Sel2_i \oplus Sel4_i) * P_Y \oplus Sel3_i * P_{3Y}\\
&\quad \oplus Sel2_i * Y_{n-1} \oplus N_i \oplus 1. \qquad (11)
\end{aligned}$$

Equation (10) shows the parity of the i-th partial product (excluded are the first and last ones). Here the first term computes the parity of bits of lower weight. The following terms represent the two most significant bits (the last one is inverted). The last summand N_i is the signal added to the LSB. That equation simplifies to (11).

The first partial products parity is given by

$$\begin{aligned} &P_{\mathrm{Part}_{i=0}} \\ &= (Sel1_0 \oplus Sel2_0 \oplus Sel4_0) * P_Y \\ &\quad \oplus (Sel1_0 \oplus Sel4_0) * Y_{n-1} \\ &\quad \oplus Sel3_0 * (P_{3Y} \oplus (3Y)_{n+1}) \oplus 1. \end{aligned} \tag{12}$$

If $n \equiv 1 \bmod 3$, as in 16- or 64-bit multipliers, the parity of the last partial product is given by

$$\begin{aligned} &P_{\mathrm{Part}_L} \\ &= \sum_{j=0}^{n-1} (Sel1_L * Y_j \oplus N_L) \oplus 1 \oplus 1 \oplus N_L \\ &= Sel1_L * P_Y \oplus N_L. \end{aligned} \tag{13}$$

In 8- and 32-bit multipliers that parity slightly differs from this equation. We have

The parity of all partial products for $n \equiv 1 \bmod 3$ is computed by

$$\begin{aligned} &P_{\mathrm{Part}} \\ &= P_{\mathrm{Part}_0} + P_{\mathrm{Part}_L} + \sum_{i=1}^{L-1} P_{\mathrm{Part}_i} \\ &= (P_{Sel1} \oplus P_{Sel2} \oplus P_{Sel4}) * P_Y \\ &\quad \oplus P_{Sel3} * P_{3Y} \oplus Sel3_0 * (3Y)_{n+1} \\ &\quad \oplus (P_{Sel2} \oplus Sel1_0 \oplus Sel2_0 \oplus Sel4_0) * Y_{n-1} \\ &\quad \oplus P_N \oplus N_0 \oplus 1. \end{aligned} \tag{14}$$

It depends on the number of partial products; the remaining ones in the partial products are added to one or zero (the last term in (14)). So (14) is valid for $n = 16$ or 64.

5. Experimental results

The multiplier was implemented in VHDL, tested by ModelSim and mapped to the vtvtlib25 cmos library (Sulistyo and Ha, 2002, 2003) by the Synopsis design compiler. Optimization was performed for the area and delay, such that all multipliers have almost the same delay. Results for the operand sizes 8, 16, 32 and 64 are given in Table 1.

By increasing operand sizes, Booth-3 becomes smaller than Booth-2. For 64 bits the method needs 88% of the area of Booth-2.

For 64 bits the proposed self-checking Booth-3 multiplier only needs 102% of the area of Booth-2 without error detection and 115% of Booth-3 without error detection. For the same configuration, the proposed self-checking Booth-2 multiplier needs 122% of Booth-2 without error detection. Thus the higher performance for Booth-3 in the linear carry-save-a dder is preserved and slightly strengthened.

Table 3. Booth-3-decoder.

X	N	$Sel_{1,2,3,4}$	Sel_0
0000	0	0000	1
0001	0	1000	0
0010	0	1000	0
0011	0	0100	0
0100	0	0100	0
0101	0	0010	0
0110	0	0010	0
0111	0	0001	0
1000	1	0001	0
1001	1	0010	0
1010	1	0010	0
1011	1	0100	0
1100	1	0100	0
1101	1	1000	0
1110	1	1000	0
1111	1	0000	1

The Booth-2 multiplier presented in (Nicolaidis and Duarte, 1998) and extended by one parity tree has an overhead of 142%. Thus, a significant cost reduction in comparison with that multiplier could be achieved. We emphasize that the small extra parity tree was inserted to check the parity of Y (the parity of X can be checked by some two-rail-checker outputs), to meet the code-disjoint property and to make all multipliers comparable.

6. Conclusions

An efficient self-checking code-disjoint Booth-3 multiplier was proposed based on a linear CSA and a final CRA. The multiplier detects single input faults, single stuck-at-faults at logical gates and all errors in an output register. An error detection scheme without duplicating decoder cells was presented and combined with parity prediction. The code-disjoint-property could be combined with forming partial products and no extra parity tree is needed. Parity prediction for Booth-2 and -3 multipliers was developed using the standard sign-extension.

Acknowledgments

The authors would like to thank Prof. Michael Goessel from the Fault Tolerant Computing Group at the University of Potsdam for stimulating discussions.

References

Al-Twaijry H. A. and Flynn M. J. (1995). Performance/area tradeoffs in Booth multipliers, *Technical Report CSL-TR-95-684*, Stanford University.

Booth A. D. (1951). A signed binary multiplication technique, *The Quarterly Journal of Mechanics and Applied Mathematics* **4**(2): 236–240.

Goessel M. and Graf F. (1993). *Error Detection Circuits*, McGraw-Hill, London.

Gorshe S. S. and Bose B. (1996). A self-checking ALU design with efficient codes, *Proceedings of the 14th IEEE VLSI Test Symposium (VTS '96)*, IEEE Computer Society, Washington, DC, USA, p. 157.

Hsiao M. and Sellers F. (1963). The carry dependent sum adder, *IEEE Transactions on Electronic Computers*, **EC-12**: 265–268.

Hunger M. (2006). Self-checking Booth-3 multiplier, *Proceedings of the 1st International Conference for Young Researchers in Computer Science, Control, Electrical Engineering and Telecommunications*, Zielona Góra, Poland.

Lala P. (Ed.) (2001). *Self-Checking and Fault-Tolerant Tigital Tesign*, Morgan Kaufmann Publishers Inc., San Francisco, CA.

Lo J. C., Thanawastien S. and Rao T. R. N. (1993). Berger check prediction for array multipliers and array dividers, *IEEE Transactions on Computers* **42**(7): 892–896.

Marienfeld D., Sogomonyan E. S., Ocheretnij V. and Gossel M. (2004). A new self-checking multiplier by use of a code-disjoint sum-bit duplicated adder, *Proceedings of the 9th IEEE European Test Symposium (ETS'04)*, IEEE Computer Society, Washington, DC, USA, pp. 30–35.

Marienfeld D., Sogomonyan E. S., Ocheretnij V. and Gossel M. (2005). New self-checking output-duplicated booth multiplier with high fault coverage for soft errors, *ATS '05: Proceedings of the 14th Asian Test Symposium,* IEEE Computer Society, Los Alamitos, CA, USA, pp. 76–81.

Nicolaidis M. and Duarte R. O. (1998). Design of fault-secure parity-prediction booth multipliers, *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '98)*, IEEE Computer Society, Washington, DC, USA, pp. 7–14.

Nicolaidis M., Duarte R. O., Manich S. and Figueras, J. (1997). Fault-secure parity prediction arithmetic operators, *IEEE Design and Test* **14**(2): 60–71.

Ocheretnij V., Sogomonya E. S. and Goessel M. (2001). A new code-disjoint sum-bit duplicated carry look-ahead adder for parity codes, *Proceedings of the 10th Asian Test Symposium (ATS '01)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 365.

Parhami B. (2001). *Instructor´s manual for "Computer Arithmetic: Algorithms and Hardware Designs"*, Vol. 2: Presentation Material, Oxford University Press, Oxford.

Shivakumar P., Keckler S. W., Kistler M., Burger D. and Alvisi L. (2002). Modeling the effect of technology trends on the soft error rate of combinatorial logic, *Proceedings of the International Conference on Dependable Systems and Networks,* pp. 389–398.

Sparmann U. and Reddy S. M. (1994). On the effectiveness of residue code checking for parallel two's complement multipliers, *Proceedings of the 24th International Symposium on Fault Tolerant Computing FTCS-24*, IEEE Computer Society Press, Austin, TX, USA, pp. 219–229.

Sulistyo J. B. and Ha D. S. (2002). A new characterization method for delay and power dissipation of standard cells, *VLSI Design* **15**(3): 667–678.

Received: 25 January 2007
Revised: 26 May 2007
Re-revised: 9 May 2008