Fault Localization, Error Correction, and Graceful Degradation in Radix 2 Signed Digit-Based Adders

Gian Carlo Cardarilli, *Member*, *IEEE*, Marco Ottavi, *Member*, *IEEE*, Salvatore Pontarelli, Marco Re, *Member*, *IEEE*, and Adelio Salsano, *Member*, *IEEE*

Abstract—In this paper, a methodology for the development of fault-tolerant adders based on the Radix 2 Signed Digit (SD) representation is presented. The use of a number representation characterized by a carry propagation confined to neighbor digits implies interesting advantages in terms of error detection, fault localization, and repair. Errors caused by faults belonging to a considered stuck-at fault set can be detected by a parity-based technique. In fact, a carry-free adder preserving the parity of the augends can be implemented allowing fault detection by using a parity checker. Regarding fault localization, the "carry-free" property of the adder ensures the confinement of the error due to a permanent fault to only few digits. The detection of the faulty digit has been obtained by using a recomputation with shifted operands method. Finally, after the fault localization, graceful degradation of the system intended as the reduction of the performances versus a correct output computation can be obtained by using two different procedures. The first one allows obtaining the correct output by recomputing the result performing two different shift operations and using the intersection of the obtained results to recover the correct output, while the second one is based on a reduced dynamic range approach, which allows us to obtain the result in only one step, but with fewer output digits.

Index Terms—Fault tolerance, high-speed arithmetic, error checking.

1 Introduction

The increasing scaling rate of the microelectronic technologies observed in recent years pushes for the use of fault-tolerant techniques, traditionally used in high reliability applications such as aerospace and avionics and also in commercial applications. For example, the effects of neutrons at sea levels when subnanometric microelectronics technologies are used seems to be not negligible. In fact, as semiconductor technology advances, the amount of charge that is stored in specific nodes in the circuitry continues to decline [1]. Many vendors, such as Xilinx, are starting to furnish special tools and architectures to face these problems in the FPGA case [2]. Due to these facts, since adders are essential building blocks in all data processing systems, the design of arithmetic structures with online error detection and correction capabilities represents an important research topic. In the literature, a number of self-checking adder implementations have been proposed, such as based on residue codes [3], [4], parity codes, [5], [6], [7], or Berger codes [8]. Other error detection techniques are based on recomputing with shifted [9] and/or rotated operands, as shown in [10], [11].

Other solutions based on carry-free self-checking adders have been proposed in the literature such as in [12], [13]. In particular, in [12], an inherent parity coding scheme to code the digits is proposed, while, in [13], a one out of three scheme is investigated. Besides, not many works propose adders which provide combined error detection and correction capabilities. The most widely applied techniques to obtain error correction in adder circuits are based on time-redundancy [11] or on the residue number system representation [14], [15].

The objective of this paper is the design of a self-checking adder architecture by combining parity checking techniques and SD number representation. The parity coding scheme proposed in this paper [16] allows implementing a self-checking adder with fault localization and graceful degradation capabilities. The main idea is that, in SD representation, the carry propagation is limited only to the neighbor digits, allowing us to set up a procedure to locate the faulty digits by means of ad hoc algorithms. Moreover, the locality of the carry propagation allows us to obtain graceful degradation. The paper is organized as follows: In Section 2, a brief overview of the SD arithmetic is reported, while, in Section 3, the chosen digit coding is reported and discussed with respect to fault detection. In Section 4, the self-checking adder architecture is proposed, together with an analysis of its implementation overhead. Section 5 reports the procedure to obtain fault localization and shows the possible graceful degradation approaches. Finally, in Section 6, the conclusions are drawn.

2 Background

The general theory and application of the SD representation is reported in [17], [18]. In this section, a brief illustration of

• *The authors are with the Department of Electronic Engineering, University of Rome "Tor Vergata," Via Del Politecnico 1, 00133, Rome, Italy. E-mail: {g.cardarilli, marco.re}@ieee.org, {ottavi, pontarelli, salsano}@ing.uniroma2.it.*

Manuscript received 20 July 2004; revised 3 June 2005; accepted 30 Aug. 2005; published online 22 Mar. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0242-0704.

TABLE 1
ADD1 Functions

Addend, Augend Digits Position i	Sign Info on Digits Position $i-1$	Intermediate Carry Digit c_i	Intermediate Sum Digit w_i
-1,-1	Not Used	-1	0
-1,0	$(a_{i-1}+b_{i-1})<0$	-1	1
-1,0	Otherwise	0	-1
0,0	Not Used	0	0
1,-1	Not Used	0	0
1,0	$(a_{i-1}+b_{i-1})>0$	1	-1
1,0	Otherwise	0	1
1,1	Not Used	1	0

its basic theory is shown. In a radix r SD representation, a number x can be represented as

$$x=\sum_{i=0}^{n-1} x_i r^I, \tag{1}$$

where the digit set is $x_i \in \{-a,\ldots,-1,0,1,\ldots,a\}$, with $\lceil \frac{r-1}{2} \rceil \leq a \leq r-1$.

The original motivation for introducing SD representation was to eliminate the carry propagation chains in addition and subtraction [19]. In fact, given two operands x and y, the addition operation can be split into the two operations

$$w_i = x_i + y_i - rc_i, \tag{2}$$

$$z_i = w_i + c_{i-1}, \tag{3}$$

where

$$c_i = \begin{cases} 1 \; if \; (x_i+y_i) \geq a \\ -1 \; if \; (x_i+y_i) \leq -a \\ 0 \; if \; |x_i+y_i| < a, \end{cases}$$

with w_i being an auxiliary variable. For $r=2$, there is only one possible digit set: $\{-1,0,1\}$, i.e., a must be equal to 1. Even if the condition $a \geq \lceil \frac{r-1}{2} \rceil$ cannot be satisfied, the sum can still be performed without carry propagation by using the modified rules for radix 2 SD addition proposed in [20] and reported in Table 1.

This representation allows using an architecture such as described in [17], [16] to implement a carry-free adder. The main elements of the adder are the so-called blocks ADD1 and ADD2, where ADD1 implements (2), providing the intermediate outputs c_i and w_i, while ADD2 implements (3).

3 Parity in the Carry-Free Sum

The radix-2 SD digit x_i can assume three values, hence its binary representation requires two bits. A convenient coding choice is considering digit 0 represented by either bits 00 or 11 and digit 1 and -1 represented by bits 01 and 10, respectively. This coding has two advantages:

1. Conversion from binary to SD representation is straightforward as the LSB has the same value in both representations while the MSB is put to 0 in the conversion.
2. With the proposed coding, only the MSBs of a_{i-1}, b_{i-1} are necessary and the ADD1 circuit can be implemented with 6 bits of input instead of 8. In fact, the function performed by ADD1 on a_i, b_i needs to evaluate the sign of the operands a_{i-1}, b_{i-1} in order to determine the outputs of ADD1.

As in a conventional number representation [7], with the SD representation, the parity properties of the arithmetic operations can also be used to check the correctness of arithmetic results. To define the parity of a digit, we refer to its binary coding. Thus we define the parity $\mathrm{P}(a_i)$ as the XOR of the bits representing the digit a_i and the parity of a SD number P(A) as the XOR of the parity $\mathrm{P}(a_i)$ of all digits a_i. Starting from these definitions for the parity of an SD number and assuming that P(C) and P(W) are the parities of carry and partial sum and P(Z) is the parity of the result, the following property holds:

Property 1.

$$P(Z) = P(C) \oplus P(W). \tag{4}$$

In fact, evaluating the parity of w_i and c_{i-1} with the chosen coding, we have that $P(z_i) = P(w_i) \oplus P(c_{i-1})$ for any possible combination of w_i and c_{i-1} [16]. Since the property is true for every z_i, it is true also for Z. In fact, due to the associative property of the XOR operator, the following statement holds:

$$\begin{aligned} P(Z) &= \bigoplus_{i=0}^{n-1} P(z_i) = \bigoplus_{i=0}^{n-1} (P(w_i) \oplus P(c_{i-1})) \\ &= \bigoplus_{i=0}^{n-1} P(w_i) \oplus \bigoplus_{i=0}^{n-1} P(c_{i-1}) = P(W) \oplus P(C). \end{aligned} \tag{5}$$

This demonstrates Property 1.

With a similar procedure, the following Property 2 can be demonstrated [16]:

Property 2.

$$P(W) = P(A) \oplus P(B). \tag{6}$$

In fact, for any possible combination of a_i and b_i, we have $P(w_i) = P(a_i) \oplus P(b_i)$.

4 Self-Checking Implementation of the SD Adder

In this section, the self-checking implementation of the SD adder is presented. The implementation results have been obtained by using Synopsys Design Compiler [21] and the standard cell library provided by Mississippi State University [22]. The architecture of the self-checking adder is shown in Fig. 1 and it is composed of the following blocks:

1. **Parity Prediction**, generates the value of $P(C)$;
2. **Error Indicator 1**, checks (6) and issues an error signal in case of a mismatch;

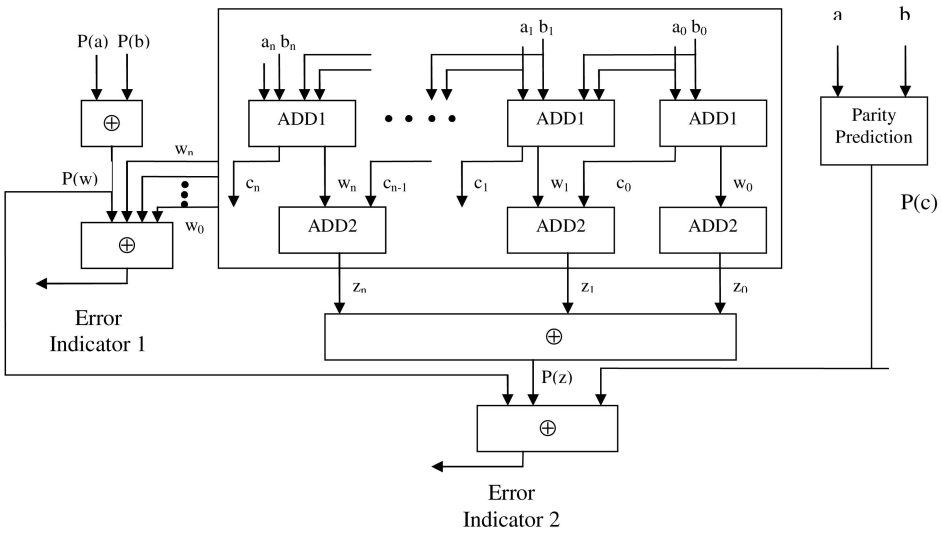

Fig. 1. Self-checking adder implementation.

3. **Error Indicator 2**, checks (4) and issues an error signal in case of a mismatch;

in addition to the standard ADD1 and ADD2 blocks used to implement an SD adder.

The Parity Prediction block is implemented by performing the XOR operation on all $P(c_i)$. The computation of c_i (and then of its parity) depends on the value of six variables. The Boolean function $P(c_i) = f[a_i(1), a_i(0), b_i(1), b_i(0), a_{i-1}(1), b_{i-1}(1)]$ for computing $P(c_i)$ has been obtained from Table 1. To detect a fault using a parity checker, we must avoid having an erroneous result $\bar{x}_i$ that has the same parity as the correct one x_i. With the chosen coding scheme, this event occurs only if the result x_i changes from -1 (01) to 1 (10) or vice versa. It can be noticed that the event of a change of the binary representation from 00 to 11 or vice versa does not introduce an error in the output value as 00 and 11 represent the same value 0. Moreover, with the chosen coding, the stuck-at in the input/output of a block can only change the value of the input digit from 0 to ± 1 or vice versa. The assumption that at least one of the parities of W and C changes when a fault occurs allows us to detect the fault by implementing (4) and (6). This assumption can be guaranteed with a suitable implementation of the blocks ADD1 and ADD2. In fact, the effect of faults inside the blocks ADD1 and ADD2 is strongly technology-dependent and, therefore, the standard cell implementation must be correctly analyzed to predict the behavior of these blocks in case of faults. First of all, it can be noticed that a fault inside an ADD1 or ADD2 block can modify more than one bit of the same digit and, in particular, can modify the value of a digit from -1 to 1 or vice versa. This case must be avoided because the parity of 1 and -1 is the same and, consequently, this kind of error cannot be detected by using a parity checker. This case occurs when a fault affects a logic cell with a fan-out greater than one.

In the literature [23], [24], different results on obtaining parity checking capability for circuits with logic resource sharing have been presented. Starting from these results, some simple considerations can be made. For the blocks ADD1 and ADD2, the resource sharing for the different bits of the same digit has been avoided. For the ADD1 block, this choice implies that the block must be split into two blocks, one providing the MSB of c_i and w_i, the other providing the LSB of c_i and w_i. The resource sharing is allowed inside each block, while it is not allowed between different blocks. With this choice, a fault inside one of the two blocks composing ADD1 can change the parity of c_i, of w_i, or of both the digits and therefore can always be detected by the parity checker. The ADD2 block provides only the digit z_i; therefore, it is synthesized as two independent blocks providing the MSB and the LSB of z_i.

To evaluate the introduced area overhead, Synopsys Design Compiler has been used for the synthesis and the results are reported in Table 2.

To avoid the occurrence of undetectable faults, ADD1 and ADD2 blocks have been synthesized by using partial

TABLE 2
Logic Cell Count

STANDARD SD ADDER	
Block	**# Logic Cell**
ADD1	23N
ADD2	11N
PROPOSED SELF CHECKING ADDER	
Block	**# Logic Cell**
ADD1 with partial sharing	30N
ADD2 with partial sharing	12N
$P(c_i)$	8N
Xor P(C)	N-1
Error Indicator 2	1
Error Indicator 1	2N-1
Xor P(W)	1
Xor P(Z)	2N-1

TABLE 3
Self-Checking Adders Comparison

	Carry Ripple	Carry Lookahead	Signed Digit
Area	$O(N)$	$O(Nlog_2N)$	$O(N)$
Time	$O(N)$	$O(log_2N)$	$O(1)$
Overhead	30%	20%	60%

sharing. This technique introduces an overhead on nine logic cells with respect to a non-fault-tolerant implementation. Besides, Error Indicator 1 and Error Indicator 2 are implemented as a two-rail parity checker, while XOR trees are used to generate $P(C)$, $P(Z)$, and $P(W)$. From the synthesis results, an N digits adder without redundancy has an area occupation in terms of logic cells of 34N, while, for the self-checking adder, we have 55N-1 logic cells. Therefore, the overhead introduced is smaller than 60 percent of the adder area. A comparison with the results obtained with the self-checking adders given in [7] and assuming a Carry Lookahead structure as proposed in [25] is reported in Table 3.

In this table, the first row reports the area occupation of the considered adders without self-checking capabilities as a function of the number of digits N, while the second row reports the required computation time. The area occupation of the self-checking SD adder increases linearly, while, in the carry lookahead case, it increases as $N\ log_2N$; therefore, for a high number of digits ($N > 16$) the area occupation of the two solutions becomes comparable, while the timing performances of the SD adder are always better.

5 Fault Localization Procedure

In this section, the fault localization procedure for the self-checking SD adder is illustrated. It is based on the recomputation with shifted operands method presented in [9]. To improve the clarity of the exposition without loss of generality, we take as an example an 8-digit adder. To correctly localize the faults inside the SD adder, we must divide the faults into three types, depending on the number of outputs affected by the fault. We remark that this classification depends on the implementation described in the previous section. In fact, each fault can affect a different number of outputs, depending on the location of the fault and on the paths from the failure point to the output ports.

1. Stuck-at fault on the ADD2 output or in one ADD1 output: The error produced by the fault is limited only to the bits of the binary representation of z_i.
2. Stuck-at in the LSB of an input digit: The fault is limited to only one ADD1 block and leads to the modification of the parity of both w_i and c_i with respect to the correct ones.
3. Stuck-at in the MSB of an input digit: A fault on an input of ADD1 can modify the value of four output digits. A stuck-at on a line of a_i can change the value of both w_i, c_i, and w_{i+1} c_{i+1}.

Regarding the bits of weight $i+1$, it can be seen that the only changes that can be introduced from an error on bits of the lower level are the modification of the outputs (w_{i+1}, c_{i+1}) from (-1, 0) to (1, -1) and vice versa or from (1, 0) to (-1, 1) and vice versa. In all these cases, the parity value of w_{i+1} does not change as P(1) = P(-1). Therefore, the parity of the erroneous value $P(\bar{W})$ depends only on the parity $P(\bar{w}_i)$ of the faulty digit $\bar{w}_i$.

We define Z as the correct output and $\overline{Z}$ as the faulty output (i.e., the output when an error indicator signal is active) and define Z_{LS}, Z_{RS} as the correct outputs obtained by using the Left and Right Shifted Inputs (LSI and RSI), respectively. Finally, $\overline{Z}_{LS}$, $\overline{Z}_{RS}$ are the outputs obtained with the shifted operands when an error indicator signal is active. It must be noticed that the shifted inputs can activate the error detection again or not depending on the occurred fault. The digits composing the operands are referred to with the corresponding lowercase letter. The equality relation $z(i) = z_{LS}(i+1)$ is valid for $0 \le i \le 6$, while a similar relation $z(i) = z_{RS}(i-1)$ is valid for $3 \le i \le 8$. For the left shifted output, the equality is not valid for $z(7)$ and $z(8)$ because the most significant digits ($a(7)$ and $b(7)$) are lost with the shift operation. For the right shifted output, the equality is not valid for $z(0)$ and $z(1)$ and $z(2)$ because the less significant digits ($a(0)$ and $b(0)$) are lost and their carries can also be lost. As stated before, the reported procedures are possible because of the carry-free features of the SD adders. Once a parity error is detected, the operation is performed again with the LSI and two different cases can be considered:

1. The parity is correct, i.e., the output is Z_{LS} ***(CASE A)***.
2. The parity is wrong, i.e., the output is $\overline{Z}_{LS}$ ***(CASE B)***.

CASE A: If the error indicator does not indicate the occurrence of the fault, the new computed value Z_{LS} and the old wrong value $\overline{Z}$ satisfy the following relation: For $0 \le j \le 6$, $z_{LS}(j+1) = z(j) = \overline{z}(j)$ for any digit that is not affected by the fault. Instead, for all the faulty digits i, we have $z_{LS}(i+1) = z(i) \neq \overline{z}(i)$. This relation allows both locating the faulty digits and correcting the output values. The number of inequalities depends on the location of the fault which caused the error. In particular, a type 1 fault corresponds to one inequality, a type 2 corresponds to two inequalities, and a type 3 to three inequalities. If no difference is found between the Z_{LS} and $\overline{Z}$ for $0 \le j \le 6$, then the fault can be localized in the two most significant digits $(z(8), z(7))$. To localize and correct these faults, the operation is performed again with RSI and the digits of Z_{RS} and the digits of Z will satisfy the following relation: For $3 \le j \le 8$, $z_{RS}(j-1) = z(j) = \overline{z}(j)$ for any digit that is not affected by the fault. Instead, for all the faulty digits i, we have $z_{RS}(i-1) = z(i) \neq \overline{z}(i)$. Thus, due to the procedure followed, we can assume that an error must be detected in the second shift operation (RSI). If, again, no difference is detected, the checker is assumed to be faulty.

CASE B: If the error indicator indicates the occurrence of the fault also in Z_{LS}, then the localization procedure is as follows: For $0 \le j \le 6$, $z_{LS}(j+1) = z(j) = \overline{z}(j) = \overline{z}_{LS}(j+1)$ holds for any digit that is not affected by the fault. Instead, the digits affected by the fault produce some inequalities between z and z_{LS}. These inequalities can be generated by both an error in the computation of the original result and an error in the recomputed result. Therefore, a minimum of

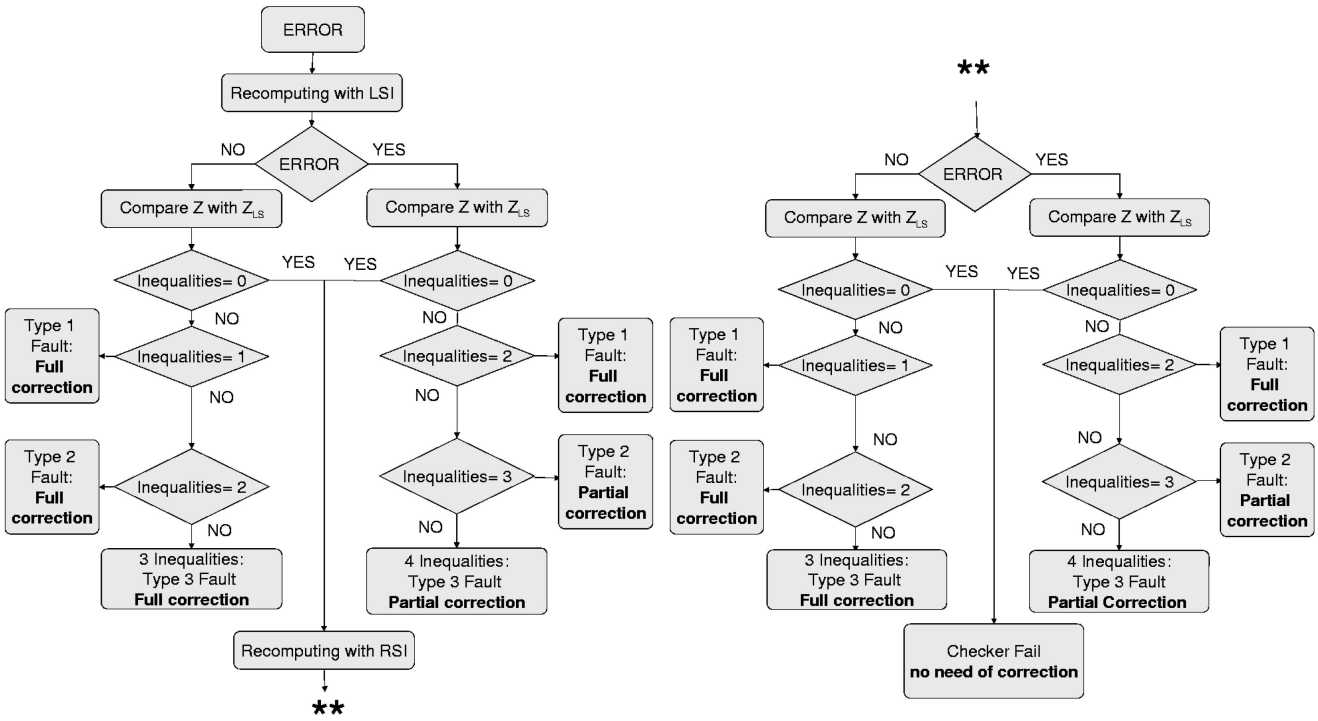

Fig. 2. Algorithm for fault detection, localization, and correction.

two and a maximum of four inequalities show up, depending on the considered type of fault of the above defined set. In particular, a type 1 fault corresponds to two inequalities, a type 2 corresponds to three inequalities, and a type 3 to four inequalities. The fault can be localized as the first different digit found by performing the comparison between $\overline{Z}$ and $\overline{Z}_{LS}$ and the type of fault can be detected by counting the number of inequalities. As in case A, if the faulty digit is the seventh or eighth or the fault is localized in the checker, no difference between Z and Z_{LS} is detected. Recomputing with RSI allows us to detect if the fault affects the remaining digits or the checker. The procedure which has been developed can also detect the faults occurring on the error indicator blocks. In fact, if an error is detected by the parity checkers, but no inequality has been observed between the original computed value and the values obtained with both RSI and LSI, then the fault affects the checker itself. Thus, the outputs are correct, but the self-checking capability has been lost. The algorithm of fault detection, localization, and correction is summarized in the graph reported in Fig. 2.

For clarity of exposition, the algorithm does not report the border cases in which the inequality of the results affects the digit in position 6. In this case, the wrong digit could be only 6 (type 1 fault) or 6, 7 (type 2), or 6, 7, 8 (type 3). This case requires the further diagnosis step which uses the RSIs. The graceful degradation capabilities of the adder are related to two main aspects of this algorithm: First of all, the algorithm always allows us to detect and localize the fault; in many cases, the correct output can be obtained by accepting a performance degradation due to the time needed to the repetition of the operation with LSI and (if needed) RSI, as shown in Fig. 2. Moreover, even in those cases when it is not possible to obtain the correct output from Z_{RS} and Z_{LS}, the fault localization allows us to use the adder with a reduced dynamic range. In fact, assuming that $n_{fd} \in \{1, 2, 3\}$ faulty digits are detected in an eight digits adder, it can still be used as an $(8 - n_{fd})$ digits adder applying suitable modifications to the input vectors A and B. For instance, if the digits $\{3, 4, 5\}$ of the adder are faulty, the five digit input vectors starting from the eight digit inputs should be set up as

$$A = \{a(4), a(3), a(2), 0, 0, a(2), a(1), a(0)\}$$

and

$$B = \{b(4), b(3), b(2), 0, 0, b(2), b(1), b(0)\},$$

while the output with reduced dynamic is: $Z = \{z(5), z(4), z(3), -, -, -, z(2), z(1), z(0)\}$ The reported example is related to a type 3 fault in the $i = 3$ digit. The digits $\{0, 2\}$ and $\{6, 8\}$ are not affected by the fault, while, in order to correctly compute the $z(3)$ output, the digits $a(2)$ and $b(2)$ are repeated in position 5 to provide the correct carry to the ADD1 block in position 6.

6 Conclusions

This paper proposes a methodology for developing fault-tolerant adders by using the SD number representation. A self-checking implementation of the SD adder is illustrated and the algorithms to achieve error correction, fault localization, and graceful degradation are proposed. The main idea is to take advantage of the confined carry propagation in SD adders. This characteristic has been used to perform an error propagation analysis and to set up localization and correction procedures. The area overhead introduced for the detection of faults in the SD adder (checker) is about 60 percent. Regarding the error correction

and graceful degradation capabilities, the proposed algorithm localizes the faulty digit(s) by means of a recomputation with the shifted operands method. After fault localization, two procedures have been proposed to provide graceful degradation of the system. The first one performs two different shift operations and uses the intersection of the obtained results to recover the correct output, while the second one is based on a reduced dynamic approach, which basically allows us to obtain the result in only one step, but with fewer output digits.

REFERENCES

[1] J.F. Ziegler, "Electronic Reliability—Effects of Terrestrial Cosmic Rays," special report to NASA, Nov. 2000 (unpublished).

[2] J. Fabula, A. Lesea, and J. Moore, "Neutron-Induced Soft Error Sensitivity Characterization of FPGAs," Xilinx FPGA White Paper, Nov. 2004.

[3] W.W. Peterson, "On Checking an Adder," *IBM J. Research and Development*, vol. 2, pp. 166-168, Apr. 1958.

[4] D. Nikolos, A.M. Paschalis, and G. Philokyprou, "Efficient Design of Totally Self-Checking Checkers for All Low-Cost Arithmetic Codes," *IEEE Trans. Computers,* vol. 37, no. 7, pp. 807-814, July 1988.

[5] F.F. Sellers, M.-Y Hsiao, and L.W. Bearnson, *Error Detecting Logic for Digital Computers.* McGraw-Hill, 1968.

[6] O.N. Garcia and T.R.N. Rao, "On the Method of Checking Logical Operations," *Proc. Second Ann. Princeton Conf. Information Science System,* pp. 89-95, 1968.

[7] M. Nicolaidis, "Carry Checking/Parity Prediction Adders and ALUs," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 11, no. 1, pp. 121-128, Feb. 2003.

[8] J.-C. Lo, S. Thanawastien, T.R.N. Rao, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processors Designs," *IEEE Trans. Computer-Aided Design,* pp. 525-540, Mar. 1992.

[9] J.H. Patel and L.Y. Fung, "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands," *IEEE Trans. Computers,* vol. 31, no. 7, pp. 589-95, July 1982.

[10] J. Li and E. Swartzlander, "Concurrent Error Detection in ALUs by Recomputing with Rotated Operands," *Proc. IEEE Int'l Workshop Defect and Fault Tolerance in VLSI Systems,* pp. 109-116, Nov. 1992.

[11] M. Alderighi, S. D'Angelo, C. Metra, and G.R. Sechi, "Achieving Fault-Tolerance by Shifted and Rotated Operands in TMR Non-Diverse ALUs," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 155-163, Oct. 2000.

[12] M.A. Thornton, "Signed Binary Addition Circuitry with Inherent Even Parity Outputs," *IEEE Trans. Computers,* vol. 46, no. 7, pp. 811-816, July 1997.

[13] W.J. Townsend, M.A. Thornton, and P.K. Lala, "On-Line Error Detection in a Carry-Free Adder," *Proc. 11th IEEE/ACM Int'l Workshop Logic & Synthesis,* pp. 251-254, June 2002.

[14] L.-L. Yang, L. Hanzo, "Redundant Residue Number System Based Error Correction Codes," *Proc. Vehicular Technology Conf.*, vol. 3, pp. 1472-1476, Oct. 2001.

[15] H. Krishna, K.-Y. Lin, and J.-D. Sun, "A Coding Theory Approach to Error Control in Redundant Residue Number Systems. I. Theory and Single Error Correction," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing,* vol. 39, no. 1, pp. 8-17, Jan. 1992.

[16] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Error Detection in Signed Digit Arithmetic Circuit with Parity Checker," *Proc. 18th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems,* pp. 401-408, Nov. 2003.

[17] I. Koren, *Computer Arithmetic Algorithms.* Prentice Hall, 1993.

[18] M. Ercegovac and T. Lang, *Digital Arithmetic.* Morgan Kaufman, 2004.

[19] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel Arithmetic," *IRE Trans. Electronic Computers,* vol. 10, pp. 389-400, 1961.

[20] N. Takagi, H. Yasuura, and S. Yajima, "High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Trans. Computers,* vol. 34, no. 9, pp. 789-796, Sept 1985.

[21] http://www.synopsys.com/, 2006.

[22] http://www.erc.msstate.edu/, 2006.

[23] C. Bolchini, F. Salice, and D. Sciuto, "Parity Bit Code: Achieving a Complete Fault Coverage in the Design of TSC Combinational Networks," *Proc. Seventh Great Lakes Symp. VLSI,* pp. 32-37, Mar. 1997.

[24] N.A. Touba and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 16, no. 7, pp. 783-789, July 1997.

[25] R. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers,* vol. 31, no. 3, pp. 260-264, Mar. 1982.

Gian Carlo Cardarilli received the Laurea degree (summa cum laude) in 1981 from the University of Rome "La Sapienza." He has worked for the University of Rome "Tor Vergata" since 1984, where he is currently a full professor of digital electronics and electronics for communication systems. From 1992-1994, he worked for the University of L'Aquila. During 1987/1988, he worked for the Circuits and Systems team at EPFL of Lausanne, Switzerland. His interests are in the area of VLSI architectures for signal processing and IC design. In this field, he has published more than 140 papers in international journals and conferences. He also participated in the work group of JESSI-SMI for the support of medium and small industries. For this group, he consulted with different SMIs, designing a number of ASICs, in order to introduce microelectronics technology in the industry's products. He has also regularly cooperated with companies such as Alenia Aerospazio, Rome; STM, Agrate Brianza, Italy; Micron, Avezzano, Italy; Ericsson Lab, Rome; and with a number of SMEs. His research interests are concerned with the design of special architectures for signal processing. In particular, he works in the field of computer arithmetic and its application to the design of fast signal digital processors. He also developed mixed-signal neural network architectures implementing them in silicon technology. Recently, he also proposed different new solutions for the implementation of fault-tolerant architectures. He is a member of the IEEE.

Marco Ottavi received the Laurea degree in electronic engineering from the University of Rome "La Sapienza" in 1999 and the PhD degree in microelectronics and telecommunications from the University of Rome "Tor Vergata" in 2004. He is currently a postdoctoral research associate in the Electrical and Computer Engineering (ECE) Department of Northeastern University in Boston. In 2000, he was with the ULISSE Consortium, Rome, as a designer of digital systems for space applications. In 2003, he was with ECE Department of Northeastern University as a visiting research assistant. His research interests include yield and reliability modeling, fault-tolerant architectures, online testing, and design of nano scale circuits and systems. He is a member of the IEEE.

Salvatore Pontarelli received the Laurea degree in electronic engineering from the University of Bologna in 1999 and the PhD degree in microelectronics and telecommunications engineering from the University of Rome "Tor Vergata" in 2003. Currently, he holds a postdoctoral fellowship with the Department of Electronic Engineering of the University of Rome "Tor Vergata." His research mainly focuses on fault tolerance, online testing, and reconfigurable digital architectures.

Marco Re received the Laurea degree in electronic engineering from the University of Rome "La Sapienza" in 1991 and the PhD degree in microelectronics and telecommunications engineering from the University of Rome "Tor Vergata" in 1996. In 1998, he joined the Department of Electronic Engineering of the University of Rome "Tor Vergata" as a researcher. He was awarded two one-year NATO fellowships with the University of California at Berkeley in 1997 and 1998. His main interests and activities are in the area of DSP algorithms, fast DSP architectures, fuzzy logic hardware architectures, hardware-software codesign, number theory with particular emphasis on residue number system, computer arithmetic, and CAD tools for DSP, fault-tolerant, and self-checking circuits. He has authored or coauthored more than 80 papers. He is a member of the IEEE.

Adelio Salsano is currently a full professor of microelectronics at the University of Rome, "Tor Vergata," where he teaches courses on microelectronics and electronic programmable systems. His present research work focuses on techniques for the design of VLSI circuits, considering both CAD problems and the architectures for ASIC design. In particular, of relevant interest are the research activities on fault-tolerant/fail-safe systems for critical environments such as space, automotive, etc., on low-power systems considering the circuit and architectural points of view, and on fuzzy and neural systems for pattern recognition. An international patent and more than 90 papers in international journals or presented at international meetings are the results of his research activity. At present, he is the president of a national consortium named U.L.I.S.S.E., among 10 universities, three polytechnics, and several of the biggest national industries, such as STMicroelectronics, ESAOTE, FINMECCANICA. He is responsible for contracts with the ASI, Italian Space Agency, for the evaluation and use in the space environment of COTS circuits and for the definition of new suitable architectures for space applications. He is also involved in professional activities in the field of information technology and is also a consultant for many public authorities for specific problems. In particular, he is a consultant for the Departments of Research and of Industry, of IMI, and of other authorities for the evaluation of industrial public and private research projects. Professor Salsano was a member of the consulting Committee for Engineering Sciences of the CNR (National Research Council) from 1981 to 1994 and participated in the design of public research programs in the fields of "Telematics," "Telemedicine," "Office Automation," "Telecommunication," and, recently, "Microelectronics and Bioelectronics." He is a member of the IEEE.