A Signed Digit Adder with Error Correction and Graceful Degradation Capabilities

G.C. Cardarilli, M. Ottavi, S. Pontarelli, M.Re, A. Salsano
{ottavi,pontarelli,salsano}@ing.uniroma2.it
Department of Electronic Engineering
University of Rome "Tor Vergata"
Via Del Politecnico 1 00133 Rome, ITALY

Abstract

This paper proposes a methodology to obtain fault localization and graceful degradation of a self-checking adder based on signed digit representation. The main idea underlying the paper is to exploit the fact that in signed digit arithmetic the carry operation is confined to neighbor digits. The usage of a "carry free" adder implies some advantages in terms of error detection, fault localization and repair. For the detection standpoint a parity checker can be easily applied to detect errors caused by faults belonging to a the considered stuck-at fault set. Regarding the fault localization the "carry free" property of the adder ensures the confinement of the error due to a permanent fault only to few digits. Finally, if a fault is correctly localized, the faulty digit can be excluded and the logic which computes the other digits can be used to perform the adder operation with a reduced dynamic range.

1 Introduction

The wide diffusion of electronic systems in everyday life which has been called "ubiquitous computing" strongly relies on microcontrollers. Thus the design of these circuits oriented to on-line error detection and correction has a strong impact on the reliability of many applications and represents an important research topic. On a functional standpoint, microcontrollers can be split into two parts: data path and control path. The control path schedules the operations to be performed following the given program while the control path is composed of the arithmetic circuits which perform the actual computations. The data path of microcontrollers always includes adders, in existing literature are widely proposed self-checking adders implementations, based on residue codes as for example in [1] or on parity codes as in [2],[3],[4], Berger codes [12], or on carry free implementations as [5],[6]. The self-checking capabilities of adders with carry free implementations are widely known, while there are not many works proposing adders which provide also error correction capabilities. The most widely applied techniques to obtain error correction in adder circuits are based on time-redundancy [7] or on the residue number system representation [8], [9]. The goal of this paper is to introduce novel fault localization and graceful degradation methodology to be applied to a self checking adder [10] which uses the signed digit representation [11]. The main idea is that in signed digit representation the carry propagation is limited only to the neighbor digits and this allows to set up a procedure to locate the faulty digits by means of *ad hoc* algorithms to be implemented in the control path of the microcontroller. Moreover the locality of the carry propagation also allows to perform some extra operations in order to repair the adder to some extent by reducing its dynamic range and accepting its graceful degradation.

The paper is organized as follows: in Section 2 an overview of the characteristics of signed digit circuits is reported, while in Section 3 the procedure to obtain fault localization is proposed. In Section 4 is carried out a discussion on the different cases of fault localization reporting the proposed algorithm and showing possible graceful degradation approaches. Finally, in Section 5, the conclusions are drawn.

2 Background

In this section the basic theory of Signed Digit Representation is reported together with the description of a self-checking adder architecture. Given a number $x \in [-(2^n-1),(2^n-1)]$ it can be represented in Signed Digit representation as shown in the following:

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_0 \qquad (1)$$

Where $x_i \in \{-1,0,1\}, (i = 0,\ldots n-1)$.

Proceedings of the 10th IEEE International On-Line Testing Symposium (IOLTS'04)
0-7695-2180-0/04 $ 20.00 IEEE

This representation allows to use an architecture like described in figure 1 to implement a carry-free adder.

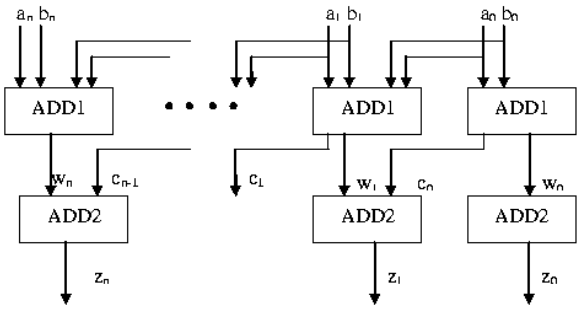

Figure 1. Adder based on Signed Digit Arithmetic

The main elements of the adder are the blocks ADD1 and ADD2 which perform the following operations.

ADD1: This block has four inputs $(a_i, b_i, a_{i-1}, b_{i-1})$ and two outputs (c_i, w_i) and performs two different operations depending on the absolute values of the operands a_i and b_i.

- when $abs(a_i) = abs(b_i)$ then $w_i = 0$ and $c_i = (a_i + b_i)div2$
- when $abs(a_i) \neq abs(b_i)$ then if $(a_i + b_i)$ and $(a_{i-1} + b_{i-1})$ have the same sign then $w_i = -(a_i + b_i)$ and $c_i = (a_i + b_i)$ otherwise $w_i = (a_i + b_i)$ and $c_i = 0$

The function implemented by the ADD1 block is summarized in Table 1.

Addends Digits Position i	Sign Info on Digits $i-1$	Carry Digit c_i	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

Table 1. ADD1 Functions

ADD2: This block has two inputs (w_i, c_{i-1}) and one output z_i and is responsible to perform the following operation:

$$z_i = w_i + c_{i-1}$$

The function implemented by the ADD2 bock is summarized in Table 2.

w_i	c_{i-1}	z_i
-1	-1	-
-1	0	-1
-1	1	0
0	-1	-1
0	0	0
0	1	1
1	-1	0
1	0	1
1	1	-

Table 2. ADD2 Function

It is always true that $2c_i + w_i = a_i + b_i$ and c_{i-1} and w_i do not have the same sign, so that $z_i \in \{-1, 0, 1\}$. The carry propagation is limited to one digit, and therefore the addition can be done in parallel, as shown in Figure 1.

bit 1	bit 0	x_i
0	0	0
0	1	1
1	0	-1
1	1	0

Table 3. Chosen coding

In [10] is shown that the adoption of a the coding reported in table 3 and the use a parity prediction block allows to implement a self-checking adder based on parity checking. The architecture of the self-checking adder is reported in figure 2. The main blocks introduced in the self checking adder are:

1. "Parity Prediction" block, generates the value of P(c)
2. Error Indicator 1, checks if $P(w) = P(a) \oplus P(b)$ and issues an error signal in case of mismatch;
3. Error Indicator 2, checks if $P(z) = P(w) \oplus P(c)$ and issues an error signal in case of mismatch;

where P(a), P(b), are the bit-to bit parity of the operands, P(w) and P(c) are referred to the intermediate values and P(z) is referred to the final result.

3 Fault localization procedure

In this section are introduced fault localization procedures for the self checking adder when stuck at faults occur in different parts of it. To improve the clarity of exposition without loss of generality, we take as example an 8 digit adder. In the following we will show how the analysis of the error behavior allows to identify the faulty digit. Depending on the fault location different types of errors can

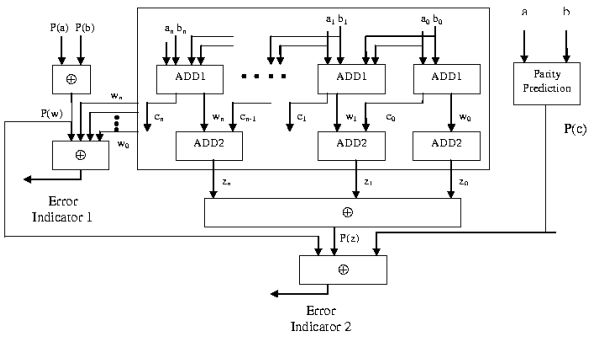

Figure 2. Self Checking adder implementation

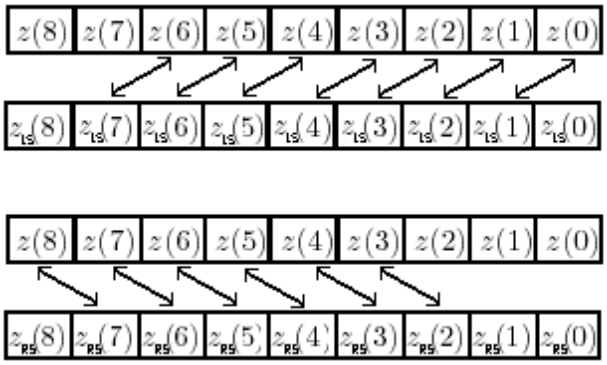

Figure 3. relation between Z, Z_{LS} and Z_{RS}

be considered and the effects on the fault localization will be shown later. The different effects of the faults cause parity errors and the consequent error indicator signals provide the information that a fault occurred. When an error is detected on the output the proposed fault localization procedure is started. In order to introduce the fault localization procedure, some definitions must be provided:

- define $A = \{a(7), a(6), a(5), a(4), a(3), a(2), a(1), a(0)\}$ and $B = \{b(7), b(6), b(5), b(4), b(3), b(2), b(1), b(0)\}$ the input operands to the adder
- define

  $Z = \{z(8), z(7), z(6), z(5), z(4), z(3), z(2), z(1), z(0)\}$ the correct output and

  $\overline{Z} = \{\overline{z}(8), \overline{z}(7), \overline{z}(6), \overline{z}(5), \overline{z}(4), \overline{z}(3), \overline{z}(2), \overline{z}(1), \overline{z}(0)\}$ the faulty output ($i.e.$ the output when an error indicator signal is active)
- define Right Shifted Inputs (RSI) the new inputs given from the the right shift of the inputs vectors such as the MSB is 0 $i.e.$ A_{RS}=$\{0, a(7), a(6), a(5), a(4), a(3), a(2), a(1)\}$ and B_{RS}=$\{0, b(7), b(6), b(5), b(4), b(3), b(2), b(1)\}$ and Left Shifted Inputs (LSI) the new inputs given from the the left shift of the inputs vectors such as the LSB is 0 $i.e.$ A_{LS}=$\{a(6), a(5), a(4), a(3), a(2), a(1), a(0), 0\}$ and B_{LS}=$\{b(6), b(5), b(4), b(3), b(2), b(1), b(0), 0\}$
- define Z_{RS}, Z_{LS} the correct outputs obtained using the shifted operands and $\overline{Z}_{RS}$,$\overline{Z}_{LS}$ the outputs obtained with the same operands when an error indicator signal is active

The relations between the original result Z and the shifted ones (Z_{LS} and Z_{RS}) are shown in Fig. 3.

As can be seen in figure the equality relation between Z and Z_{LS} is valid for $0 \leq i \leq 6$ while the same relation is valid for $2 \leq i \leq 8$ between Z and Z_{RS}. 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 also their possible carry can be lost.

It must be noted that the shifted inputs can activate again the error detection or not depending on the occurred fault. Thus even if the output of the adder is $\overline{Z}$ the result of the operation performed on the shifted operands can be Z_{RS} or Z_{LS}. The effects of four different types of faults are reported with respect to their effects on the affected digits and/or the error detection signal. As stated before, the reported procedures are possible because of the carry free features of the signed digit arithmetic adders. The faults are divided into four types depending their effects on the output digits, apart from type 4 fault which affects the checker type 1, 2, and 3 faults affect 1, 2 3 digits respectively as shown in Fig. 4.

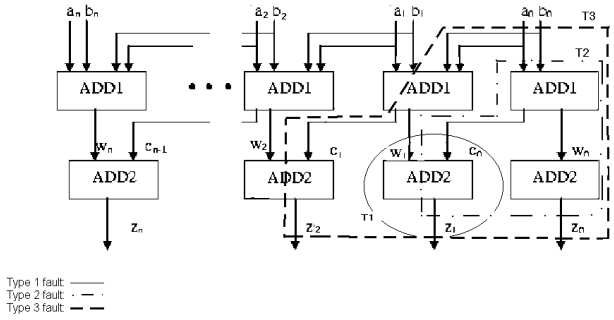

Figure 4. scope of the 3 different types of fault affecting the digits of the adder

In the following the analysis of the impact of faults on different parts of the adder is reported.

Type 1: stuck-at fault on the ADD2 output or in one ADD1 output: In this case, the error generated by the fault can affect only one digit. This error modifies the parity of Z. The chosen coding described in [10] does not allow that an error on a single bit modifies the value of the digit from -1 to 1 and vice versa (these two values have the same parity). Once a parity error is detected the operation is performed again with the LSI as stated before, 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 signal the occurrence of the fault, the new computed value Z_{LS} and the old wrong value $\overline{Z}$ have the following relation: for $0 \leq j \leq 6$ $z_{LS}(j+1) = z(j) = \overline{z}(j)$ for any digit that is not affected by the fault. Instead considering i as the faulty digit we have $z_{LS}(i+1) = z(i) \neq \overline{z}(i)$. In this case the above relation allows both to locate the faulty digit and to correct the output value. If no difference is found between the Z_{LS} and $\overline{Z}$ for $0 \leq j \leq 6$ then the fault can be localized in the two more significant digits $(z(8), z(7))$. In order to localize and correct these faults the operation is performed again with RSI inputs. In this case the digits of Z_{RS} and the digits of Z have the following relation: for $3 \leq j \leq 8$ $z_{RS}(j-1) = z(j) = \overline{z}(j)$ for any digit that is not affected by the fault. This relation shows that in the worst case, the deletion of the lowest digit can affect at most the digit with index 2. Instead, considering i as the faulty digit we have $z_{RS}(i-1) = z(i) \neq \overline{z}(i)$. Thus because of the followed procedure, we can assume that an error must be detected in the second shift operation (RSI). However, if no difference is detected again between the shifted digits and the original ones, the checker is assumed to be faulty as we will show later in the section dedicated to faults affecting the checker.

In other words the left shift operation allows to check the digits from 0 to 6 while the right shift allows to check the uppermost digits which are 7 and 8.

CASE B If the error indicator signals the occurrence of the fault also in Z_{LS}, then the localization procedure is as follows. For $0 \leq j \leq 6$ $z_{LS}(j+1) = z(j) = \overline{z}(j) = \overline{z}_{LS}(j+1)$ for any digit that is not affected by the fault. If the fault affects digit i in the interval (0-6) the following two error conditions occur:

1. $\overline{z}(i) \neq \overline{z}_{LS}(i+1)$ when $z(i) \neq \overline{z}(i)$
2. $\overline{z}(i-1) \neq \overline{z}_{LS}(i)$ when $z_{LS}(i) \neq \overline{z}_{LS}(i)$

The fault is therefore localized as the first different digit found performing the comparison between $\overline{Z}$ and $\overline{Z}_{LS}$. Moreover, the $z(i)$ digit can be corrected using the relation:

$$z(i) = z_{LS}(i+1)$$

due to the fact that with the fault localization procedure i is for sure the faulty digit and in this fault model only one digit is affected by the fault. Also in this case, if the faulty digit is the 7th or 8th, or the fault is localized in the checker, no difference between Z and Z_{LS} is detected. Using the same procedure described before, the operation can be performed again with RSI inputs in order to detect if the fault is affecting the remained digits or if it is affecting the checker.

To explain the exposed procedure we can use the following example, in which the ADD2 block computing the fourth digit is affected by a fault:

A	-55	$0\bar{1}1\bar{1}\bar{1}001$
B	97	$0111\bar{1}\bar{1}\bar{1}\bar{1}$
Z	42	$0010\bar{1}0\bar{1}\bar{1}0$
$\overline{Z}$	50	$0010\bar{1}\mathbf{1}\bar{1}\bar{1}0$

Table 4. Correct and wrong results with original inputs

A_{LS}	-110	$\bar{1}1\bar{1}\bar{1}0010$
B_{LS}	194	$111\bar{1}\bar{1}\bar{1}\bar{1}0$
Z_{LS}	84	$010\bar{1}0\bar{1}\bar{1}00$
$\overline{Z}_{LS}$	92	$010\bar{1}0\mathbf{0}\bar{1}00$

Table 5. correct and wrong results with left shifted inputs (LSI)

A_{RS}	-28	$00\bar{1}1\bar{1}\bar{1}00$
B_{RS}	49	$00111\bar{1}\bar{1}\bar{1}$
Z_{RS}	21	$00010\bar{1}0\bar{1}\bar{1}$
$\overline{Z}_{RS}$	29	$00010\mathbf{0}0\bar{1}\bar{1}$

Table 6. correct and wrong results with right shifted inputs (RSI)

The reported example considers that only one digit of the result is affected by the fault and that the error is present also in the shifted results. It can be seen that the comparison between the $\overline{z}(i)$ and $\overline{z}_{LS}(i+1)$ or between $\overline{z}(i)$ and $\overline{z}_{RS}(i-1)$ allows to localize the faulty digit and to obtain its correct value.

Type 2: stuck-at in the ADD1 block or in the LSB of a digit of one of the two inputs (A, B) : Differently from the previous case, this fault can affect one or two digits of Z, depending on the fault location and on the values of the inputs. Also in this case, the architecture proposed in [10] is able to detect the fault using the error indicator blocks. A fault localization procedure similar to the previous one can be used. In particular, the recomputation with the left

shifted inputs (A_{LS}, B_{LS}) can activate again the error detection signal, or not. Thus also Type 2 faults the *CASE A* and the *CASE B* show up.

In the first case (*CASE A*) the wrong digit can be detected comparing the Z_{LS} value with the old wrong $\overline{Z}$ value. It can be noticed that, also in this case, the faulty i digit is localized as the smaller i index such as $z_{LS}(i) \neq \overline{z}(i-1)$. The fault localization procedure can also detect if the fault is of type 1 or 2. In fact, as seen above, if the fault belongs to type 1, we obtain the equation $z_{LS}(i) \neq \overline{z}(i-1)$ only for one digit. Instead, for a type 2 fault two successive inequalities show up in the comparison of the recomputed result with the original one. Similarly to the discussion made for the type 1 faults, the comparison of the result with the one obtained using the LSI allows to locate faulty digits in the interval (0-6). Regarding the localization of faults in the (7,8) positions, the procedure which uses the RSI can be executed. The correct $z(i)$ and $z(i+1)$ digits (where i and $i+1$ are the locations of the wrong digits) can be obtained using the result of the operation performed on the shifted operands, being this result error free (in this case the error indicator is assumed not to be activated).

In *CASE B* both the original result and the recomputed one with LSI are affected by error thus we deal with $\overline{Z}$ and $\overline{Z}_{LS}$. In this case we have three inequalities between the results. The first one which is $\overline{z}_{LS}(i) \neq \overline{z}(i-1)$ is caused by the fault of the digit in position i, being $\overline{z}(i-1)$ the correct output (*i.e.* $\overline{z}(i-1) = z(i-1)$) while $\overline{z}_{LS}(i)$ is the wrong one. The second inequality $\overline{z}_{LS}(i+1) \neq \overline{z}(i)$ is related to the fault which is supposed to affect both outputs $\overline{Z}$ and $\overline{Z}_{LS}$. The third and last inequality, $\overline{z}_{LS}(i+2) \neq \overline{z}(i+1)$ is caused by the error in $\overline{z}(i+1)$ being $\overline{z}_{LS}(i+2)$ not affected by the fault in the considered carry free adder architecture. It can be noticed that, as discussed in the previous cases a similar procedure considering RSI inputs can be applied to locate the faults affecting the digits (7,8).

Type 3: stuck-at in the the MSB of an input: Also in this case the fault is detected from the error detection block. Being the MSB of input digits in position i used from ADD1 blocks in position i and $i+1$, a fault of type 3 can affect a maximum of three output digits ($z(i), z(i+1), z(i+2)$). Also with this type of fault the comparison of the $\overline{Z}$ with $\overline{Z_{LS}}$ can give rise to two cases which are related to the number of detected inequalities. In particular, *CASE A* is related to the detection of three inequalities while *CASE B* accounts for the presence of four inequalities. The localization procedure is similar to the above reported one for type 1 and 2 and also, the check of digits in position (7,8) is performed using the RSI.

Type 4: stuck-at fault in the error indicator block. This fault does not affect the correctness of the result even if the self-checking adder in its totality is affected by a fault that must be detected and localized. The procedure which has been developed in order to cope with the occurrence of the other types of faults can also detect this kind of fault. 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 we can assume that the fault affected the checker itself. Thus the outputs are correct but the self checking capability has been lost.

4 Discussion

The above reported analysis allows to implement a simple microcontroller which uses as data path the self checking ALU based on signed digit arithmetic proposed in [10] and in the control flow a specific algorithm to obtain fault localization and error correction.

The algorithm of fault detection, localization and correction is summarized in the graph reported in Figs. 5 and 6.

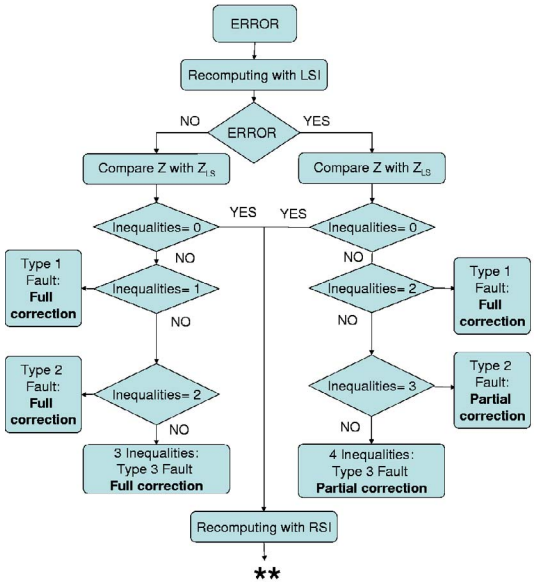

Figure 5. alghoritm using Z_{LS}

For clarity of exposition the algorithm does not report the border cases in which the inequality on the results affect 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 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 the figures. Moreover, even in those cases when

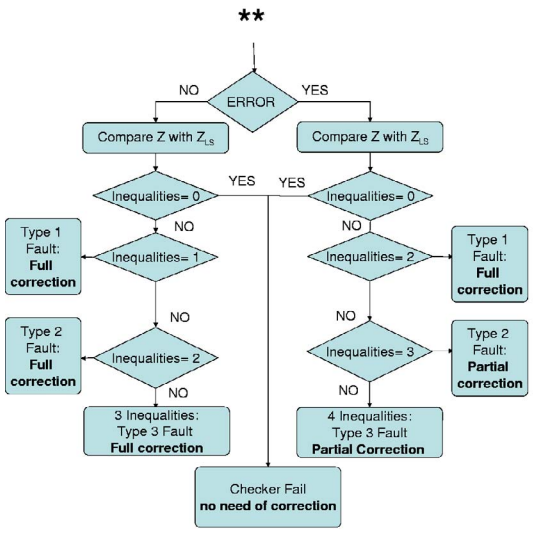

Figure 6. alghoritm using Z_{RS}

is not possible to obtain the correct output from Z_{RS} and Z_{LS}, the fault localization allows to use the adder with a reduced dynamic. In fact assuming that $x \in \{1, 2, 3\}$ faulty digits are detected a 8 digits adder, it can still be used as a $(8 - x)$ 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 8 digits inputs should be set up as follows

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

While the output with reduced dynamic is:

$$Z = \{z(5), z(4), z(3), -, -, -, z(2), z(1), z(0)\}$$

The above reported example is related to a type 3 fault in the $i = 3$ digit. Obviously the digits $\{0, 2\}$ and $\{6, 8\}$ are not affected by the fault, while in order to compute correctly 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.

5 Conclusions

In this paper a methodology to achieve error correction fault localization and some extent of graceful degradation of a self checking signed digit adder has been proposed. The main idea underlying the paper is to exploit the fact that in signed digit arithmetic the carry operation is confined to neighbor digits. This characteristic has been used to perform an error propagation analysis and to set up a localization and correction algorithm to be implemented in the control flow of a possible microprocessor which implements signed digit ALU. The proposed algorithm is able to localize the faulty digit(s) by means of a recomputation of the error affected outputs with shifted operands. After the fault localization the proposed algorithm allows to introduce some extent of graceful degradation of the system intended as the reduction of the performances of the ALU vs a correct output computation. In fact two procedures are reported. The first one allows to obtain the correct output by recomputing the result performing two different shift operations and using the intersection of the obtained data to recover the correct output. The other introduced procedure is based on a reduced dynamic approach, which basically allows to obtain the result in only one step, but with fewer digits on the output. As a concluding remark, it is important to note that no extra hardware overhead has been introduced with respect to the self-checking implementation of the adder.

References

[1] W. W. Peterson "On checking an adder" I.B.M. J. Res. Develop., vol 2, pp. 166-168, Apr 1958.

[2] F. F. Sellers, M.-Y Hsiao, and L. W. Bearnson, "Error Detecting Logic for Digital Computers" New York: McGraw-Hill, 1968.

[3] O. N. Garcia and T. R. N. Rao, "On the method of checking logical operations", in Proc. 2nd Annu. Princeton Conf. Information ScienceSystem, 1968, pp. 89-95.

[4] Nicolaidis, M. "Carry checking/parity prediction adders and ALUs" IEEE Transactions on very Large Scale Integration (VLSI) Systems, Volume: 11 Issue: 1 , Feb 2003 Page(s): 121 -128

[5] M. A. Thornton, "Signed binary addition circuitry with inherent even parity outputs" IEEE Trans. on Computers, vol. 46, pp.811-816, July 1997.

[6] W. J. Townsend, M. A. Thornton, and P. K. Lala, "On-Line Error Detection in a Carry-Free Adder", 11th IEEE/ACM International Workshop on Logic & Synthesis pp. 251-254, New Orleans, LA, June 4-7, 2002

[7] Alderighi, M.; D'Angelo, S.; Metra, C.; Sechi, G.R.; "Achieving fault-tolerance by shifted and rotated operands in TMR non-diverse ALUs" in Proceedings. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 25-27 Oct. 2000 pp. 155-163

[8] Lie-Liang Yang, Hanzo L., "Redundant residue number system based error correction codes" in Vehicular Technology Conference, 2001. VTC 2001 Fall. IEEE VTS 54th , Volume: 3 , 7-11 Oct. 2001 pp. 1472 - 1476

[9] Krishna, H.; Lin, K.-Y.; Sun, J.-D. "A coding theory approach to error control in redundant residue number systems. I. Theory and single error correction", Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, vol. 39, issue 1, Jan. 1992, pp. 8 - 17

[10] Cardarilli, G.C.; Ottavi, M.; Pontarelli, S.; Re, M.; Salsano, A., "Error detection in signed digit arithmetic circuit with parity checker" Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on , 3-5 Nov. 2003 pp. 401 - 408,

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

[12] 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.