# Error detection in Signed Digit Arithmetic circuit with parity checker

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

## Abstract

This paper proposes a methodology for the development of simple arithmetic self-checking circuits using Signed Digit representation. In particular, the architecture of an adder is reported and its self- checking capability with respect to the stuck-at fault set is shown. The main idea underlying the paper is to exploit the properties of Signed Digit representation allowing carry-free operations. In a carry free adder the parity can be easily checked allowing therefore detecting the occurrence of a fault belonging to the considered stuck- at fault set. The proposed architecture is therefore very suitable for the implementation of self-checking adders that are also fast due to the same carry free property.

## 1 Introduction

Adders are one of the building blocks of integrated circuits as they are used in ubiquitous applications like computing and controlling. Therefore the self-checking implementation of these arithmetic circuits has a strong impact on the reliability of many applications and has always represented an important research topic. Many literature proposes self-checking adders implementations either based on residue codes as for example in [2] or on parity codes as in [3]. In this paper a self checking adder architecture is proposed, the proposed architecture relies on Signed Digit representation properties [1] in order to implement an easy parity checking scheme of the operation.

Parity coding is already used in ALU and register file implementations, hence the embedding of the proposed solution can exploit already available redundancies. On the other hand the implementation of an adder in Signed Digit representation, allows obtaining fast arithmetic circuits due to the carry-free property and also can be used

in the implementation of RNS modules [6]. Therefore, the use of techniques of fault detection as described in this paper can be applied both in the implementation of high speed and highly reliable circuits either, if used with RNS, in the implementation of graceful degradable arithmetic circuits. The latter option relies on the technique of excluding the faulty RNS module and working with reduced dynamic. In previous literature other solutions based on carry-free self checking adders have been proposed [4] [5] in particular, in [4] an inherent parity coding scheme to code the digits is proposed while in [5] a 1 out of 3 scheme is investigated. In the proposed solution a different coding scheme is proposed allowing fast conversion and low overhead in the adder implementation.In the paper the proposed solution is analyzed and it is shown to be self checking with respect to faults occurring on both levels of the SD sum. The area occupancy and the overhead introduced by the checker in an FPGA implementation is also provided.

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 chosen digit coding is reported and discussed with respect to fault detection. In Section 4 the characteristics of the carry-free operation with respect to the parity coding is reported, while in Section 5 the self checking adder architecture is proposed together with an analysis of its implementation overhead. Finally, in Section 6 conclusions and future works are drawn.

# 2 Background

In this section the basic theory of Signed Digit Representation is reported together with the description of the carry free 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:

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \ldots + x_0 \tag{1}$$

Where $x_i \in \{-1, 0, 1\}, (i = 0, \ldots n-1)$. This representation allows to use an architecture like described in figure 1 to implement a carry-free adder. The main elements of the adder are the blocks ADD1 and ADD2 that 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 is responsible to perform the following operations in case the absolute values of the operands $a_i$ and $b_i$ are the same or are different.

- 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.

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

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

| 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 |

Table 1: ADD1 Functions

| $w_i$ | $c_{i-1}$ | $z_i$ |
|---|---|---|
| -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

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

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.

# 3 Parity and chosen code

From the previous description we have that each digit $x_i$ can assume three values, hence its binary representation must rely on two bits. The coding reported in the following table has been chosen.

The adoption of this coding has two advantages:

1. conversion from binary to SD representation is straightforward as the less significant bit (bit 0 in table) has the same value in both representations while the other bit (bit 1) is put to 0 in the conversion.

2. as described before, 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 output 1. With the

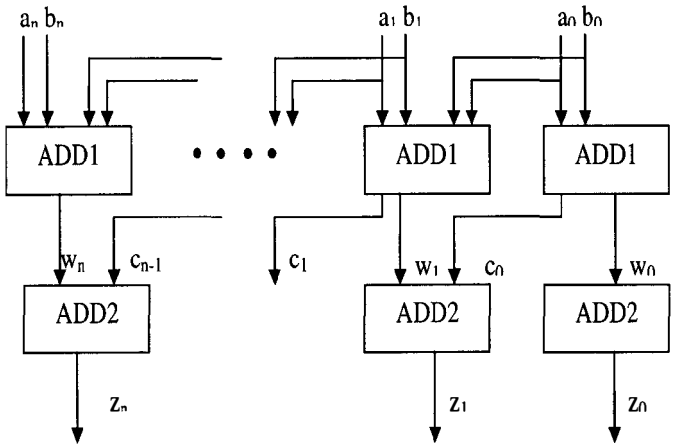


Figure 1: Adder based on Signed Digit Arithmetic

| bit 1 | bit 0 | $x_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | -1 |
| 1 | 1 | 0 |

Table 3: Chosen coding

proposed coding only the MSBs of $a_{i-1}, b_{i-1}$ are necessary and the ADD1 circuit can be implemented with 6 input bits instead of 8.

Using the reported coding any error occurring on one of the bits representing the digit $x_i$ does not invert the absolute value of the digit $x_i$ itself. In other words, an error on one of the bits representing a digit can never cause the inversion of its sign i.e. from -1 to 1 or vice versa. Defining P(a) the sum modulo 2 of all the bits representing the number $a$ we have that each error on a single bit of the representation leads to a variation of the parity P(a).

# 4 Parity in the carry-free sum

The coding reported in Table 3 allows deducing some properties of sum operation with respect to the parity calculation. Assuming P(c) and P(w) as the parities of carry and partial sum and P(z) the parity of the result, then the following properties are demonstrated:

**Property 1**

$$P(z) = P(c) \oplus P(w) \tag{2}$$

(from the definition of ADD1) It cannot happen that $c_{i-1}$ and $w_i$ have the same sign hence for every $z_i$ the values can be as reported in the following Table 4

| $z_i$ | $w_i$ | $c_{i-1}$ |
|---|---|---|
| 00(11) | 01 | 10 |
| 01 | 01 | 00 |
| 10 | 10 | 00 |
| 00(11) | 00(11) | 00(11) |

Table 4: values of $z_i$ in function of $w_i$ and $c_{i-1}$

Because the property is true for every $z_i$, it is true also for $z$.

**Property 2**

$$P(w) = P(a) \oplus P(b) \tag{3}$$

Like before, the demonstration is based on showing that $P(w_i) = P(a_i) \oplus P(b_i)$ holds for every i.

From the following table we have:

| $w_i$ | $a_i$ | $b_i$ | $P(w_i)$ | $P(a_i)$ | $P(b_i)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| $\pm 1^*$ | 0 | 1 | 1 | 0 | 1 |
| $\pm 1^*$ | 0 | -1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | -1 | 1 | 0 | 1 | 1 |
| 0 | -1 | -1 | 0 | 1 | 1 |

Table 5: values of $P(a_i)$ $P(b_i)$ and $P(w_i)$

* Sign depends on the values of $a_{i-1}$ and $b_{i-1}$
as can be seen for all the cases $P(w_i) = P(a_i) \oplus P(b_i)$, therefore $P(w) = P(a) \oplus P(b)$

# 5 Self-checking adder

Before introducing self-checking adder the behavior of the adder of Figure 1 is described in case of the occurrence of stuck-at faults in different parts of it:

1. Stuck-at on an output of ADD2:

   In this case, the error produced by the output of block ADD2 modifies the parity of z as the coding described in section 3 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). The error is detectable with a parity checker on the output.

2. Stuck-at on an input of ADD2 (i.e. output of ADD1)

   Also in this case the fault is limited to only one block and leads to the modification of the parity of result z with respect to the correct one. A particular case must be considered though, i.e. when a one of the inputs is 1 (-1) and the other changes

from 0 to 1 (-1). Even if this case is not contemplated in the normal behavior of the circuit, it is possible anyway to implement ADD2 to output 0 when such a configuration shows up at the inputs. This again allows having an output z with a different parity with respect to the correct one.

3. Stuck-at on an input of ADD1

   As can be seen in Figure 1, a fault on an input of ADD1 can modify the value of four outputs. As an example, a stuck-at fault 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 noticed 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). Instead, regarding the bits of weight $i$, an error on one of the inputs $a_i$ and $b_i$ modifies always the value of $w_i$ and its parity.

From the above analysis it can be stated that all the type 1 and 2 faults can be detected checking the parity P(z), while all the type 3 faults can be detected checking the parity P(w).

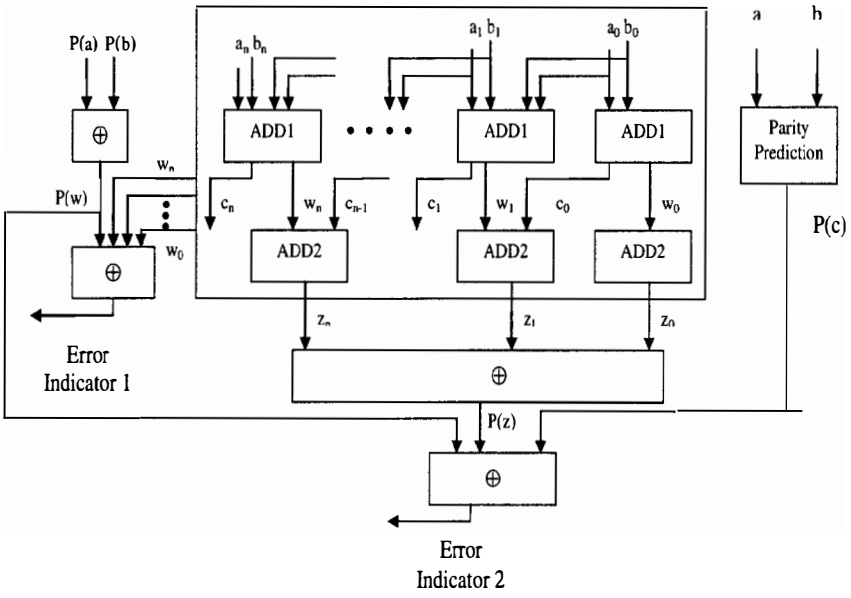


Figure 2: Self Checking adder implementation

In Figure 2 the block diagram of self-checking adder is introduced. 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;

The values of P(w) and P(z) are calculated by means of parity generating circuits. According to the above discussion, the output of Error Indicator 1 allows detecting the occurrence of type 3 faults, while the output of Error Indicator 2 allows detecting the occurrence of both type 1 and type 2 faults. The Parity Prediction block is implemented by performing the xoring of all $P(c_i)$. As seen in section 3, the computation of $c_i$ (and then of its parity), depends on the value of 6 variables. The boolean function

$$P(c_i) = f[a_i(1), a_i(0), b_i(1), b_i(0), a_{i-1}(1), b_{i-1}(1)]$$

has been easily deduced from the above reported tables. In order to evaluate the area overhead introduced by the checker an FPGA synthesis has been performed using Xilinx Virtex XCV1000 as target device. From the results output by synplify synthesis tool we obtained the results reported in Table 6.

| | **# inputs** | **# outputs** | **# 4 input LUTs** |
|---|---|---|---|
| ADD1 | 6 | 4 | 8 |
| ADD2 | 4 | 2 | 2 |
| $P(c_i)$ | 6N | N | 3N |

Table 6: I/O and LUT count

The Error Indicator 1 and Error Indicator 2 are implemented as two-rail parity checker while xor trees are used to generate P(c), P(z) and P(w). The two-rail implementation of the Error Indicator blocks allows the detection af a stuck-at fault inside this blocks, while a fault in the parity prediction blocks is detected by the Error Indicator block. The count of LUT number for these trees are reported for N=8,16,32 in the following table:

| | **N=8** | **N=16** | **N=32** |
|---|---|---|---|
| Error Indicator 1 | 5 | 11 | 21 |
| Error Indicator 2 | 1 | 1 | 1 |
| Xor P(w) | 1 | 1 | 1 |
| Xor P(c) | 3 | 5 | 11 |
| Xor P(z) | 5 | 11 | 21 |

Table 7: Blocks Overhead

Assuming to have N digits operands and given the area overhead of the adder without redundancy from $\#LUT = N * (\#LUT_{ADD1} + \#LUT_{ADD2})$ and the area occupation of the checker from the previous tables, we can draw the following table for the area overheads.

As can be seen, the overhead introduced by the checker is less than 50% of the adder area and the value slightly decreases with the increase of word length.

| | Adder | Checker | % overhead |
|---|---|---|---|
| N=8 | 80 | 39 | 49 |
| N=16 | 160 | 77 | 48 |
| N=32 | 320 | 151 | 47 |

Table 8: Checker Overhead

# 6 Conclusions and future work

In this paper the architecture of a self checking Signed Digit adder has been proposed. Exploiting the characteristic of the Signed Digit arithmetic, the reported architecture is shown to be self checking with respect to stuck-at class faults by using parity checking. The introduced coding for signed digit encoding has proved to be also effective in order to easily implement the conversion from binary coding and to obtain area effective adder as well as checker implementation. The introduced overhead of the checker is proved to be less than 50% of the adder. Future work will include the use of this adder as building block of reliable RNS arithmetic circuits as well as high speed filters.

# References


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

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

[3] 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

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

[5] 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

[6] Shugang Wei; Kensuke Shimizu, "Fast residue arithmetic multipliers based on signed-digit number system", The 8th IEEE International Conference on Electronics, Circuits and Systems, 2001. ICECS 2001. Volume: 1, 2001 Page(s): 263 -266 vol.1