ISSN 22773061

HYBRID-CMOS LOGIC STYLE DESIGN FOR FAST SELF-CHECKING ADDERS DATA PATHS

Belgacem HAMDI[1,2], Khaled BENKHALIFA[2], Aymen FRADI[1]
[1]Electronic & Microelectronics' LAB. Monastir Tunisia
Belgacem.hamdi@gmail.com, belgacem.hamdi@issatgb.rnu.tn
[2]Institute of Applied Sciences and Technology of Sousse, Tunisia
Khaled.benkhalifa@issatso.rnu.tn

ABSTRACT

In this paper we present an efficient design for self-checking fast adders data paths. We investigate the implementation of concurrent error detection fast adders: carry look-ahead, Carry skip, Carry-select and Conditional-Sum adders. To achieve a low overhead, low power design, we use hybrid-CMOS logic style and combine Conventional CMOS and CMOS Pass transistor Logic (CPL). The proposed schemes are Totally Self-Checking (TSC). They are fully differential and checked by dual-rail and parity codes.

Indexing terms/Keywords

Self-Checking adders, fast adders, Hybrid-CMOS Logic Style, differential XOR, carry look-ahead, Carry skip, Carry-select, Conditional-Sum adder.

Academic Discipline And Sub-Disciplines

Computers; Microelectronic design, Design For Test;

SUBJECT CLASSIFICATION

Reliability of Integreted Circuits; Self-Checking data paths

TYPE (METHOD/APPROACH)

Digital CMOS Design; Design Analysis; formal proof; Electrical simulations

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Vol 10, No. 6
editor@cirworld.com
www.cirworld.com, member.cirworld.com

ISSN 22773061

I. INTRODUCTION

Addition is one of the most fundamental operations for digital computations. It usually impacts widely the overall performance of digital systems and crucial arithmetic function.Thus much effort has been invested in research that has led to faster and more efficient ways to perform this operation. Carry look-ahead, carry select, and carry skip adders are largely used to reduce the carry propagation delay.

On the other hand, interest in on-line error detection continues to grow as VLSI circuits increase in complexity. The main characteristic of the on-line error detection is its ability to detect transient faults that may occur in a circuit during normal operation. On-line error detection provides an opportunity for self-diagnosis and self-correction within a circuit design.

This work deals with design of fast self-checking adders' data paths. It presents a Self-Checking (SC) fast adder based on a CMOS pass transistor differential XOR [1]. Complementary pass-transistor logic (CPL) is known to be much more power-efficient than complementary CMOS. However, the pass transistor gates generate degraded signals, which slow down signal propagation. In such situation, we have to restore degraded signals generated by CPL gates [2, 1].

The proposed schema is a design using hybrid-CMOS logic style combining Complementary pass-transistor logic (CPL) and complementary CMOS. It presents a self checking fast adder based on the use of double-rail and parity encoding to achieve the totally self checking goal.

II. BACKGROUND

II.1. Self-checking (SC) circuits

Self-checking circuits are increasingly becoming a suitable approach to the design of complex VLSI ICs, to cope with the growing difficulty of on-line and off-line testing [3]. Self-checking circuits are class of circuits in which occurrence of fault can be determined by observation of the outputs of the circuits. An important subclass of these self-checking circuits is known as totally self-checking (*TSC*) circuits.

II.2. Totally self-checking (TSC) circuits

TSC circuits are used to detect errors concurrently with normal operation. These circuits operate on encoded inputs to produce encoded outputs. TSC checkers are used to monitor the outputs to indicate error when a non-code word is detected [4]. The concept of TSC circuits was first proposed in [5], and then generalized in [6].

A totally self-checking (TSC) functional block satisfies the two following properties:

- For any valid input code word and any single fault, the circuit, either produces an invalid code word on the output, or does not produce the error on the output (fault secure property).
- Any single fault is detectable by some valid input code word (self-testing property).

A circuit (functional block and checker) is TSC if the functional block and the checker are both TSC. Fig. 1 gives the basic structure of TSC circuits.

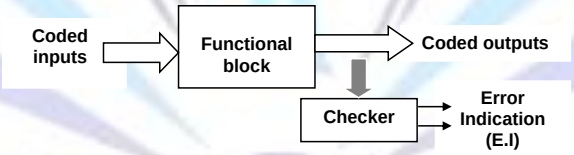

Fig 1: Basic structure of totally self-checking (TSC) circuit

The checker determines whether the output of the circuit is a valid code word or not. It also detects a fault occurring within itself [7].

Double-rail checker is based on the dual duplication code as shown in Fig. 2(a). It compares two input words X and Y that should normally be complementary ($Y = \overline{X}$) and delivers a pair of outputs coded in dual-rail.

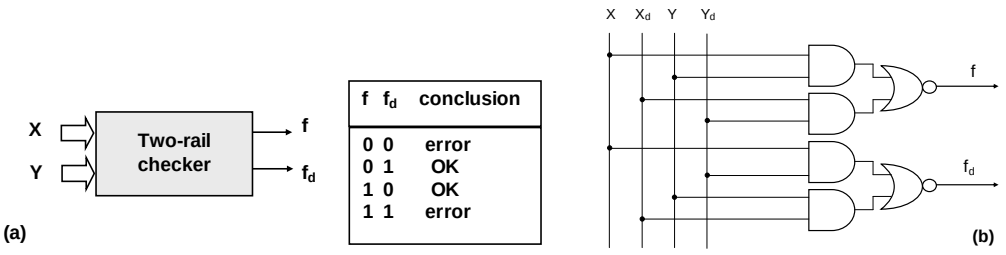

Fig 2: (a): Principle of dual-rail checker; (b): Dual-rail checker cell

ISSN 22773061

The circuit of the Fig. 2(b) is a *totally self-checking checker* since it is self-testing and code-disjoint [8].

This dual rail checker cell can be implemented in static CMOS, with 16 transistors.

II. PREVIOUS WORK

In previous work [2, 1], we proposed a self-checking fully differential full adder based on 4-transistors differential XOR gate. This scheme combine two CMOS styles: pass transistor CMOS technology to perform the sum function and static CMOS technology for the carry gate to avoid propagation problems as shown in Fig. 3. This fully differential implementation requires only 28 transistors. We also, made the proof that the proposed design is TSC (*fault secure* and *self-testing)* for all stuck-at, stuck-on and stuck-open faults.

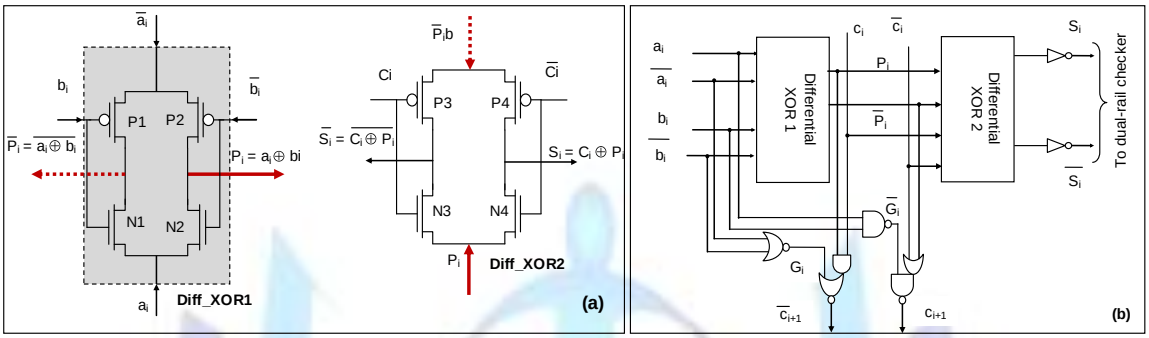

Fig 3: (a): Differential SUM Gate; (b) differential full adder

Inverters are added to restore degraded signals generated by the differential SUM gate.
Note that degraded outputs of the first gate (diff XOR1) are not used to drive the transistor gates of the second gate (diff XOR2) and thus, signals are not degraded once more. In fact, the output voltage does not depend on the number of switch transistors that the signal travels through. It only depends on the gate voltage of those switches

The proposed differential full adder delivers *propagation signals* $P_i, \overline{P_i}$ by the SUM gate, and *generation* $G_i, \overline{G_i}$ *signals* by the differential carry gate. These signals are required for the implementation of fast adders (e.g. carry look-ahead, carry skip, etc.)

IV. Fast carry propagation SC adders

In [2] a self-checking ripple-carry adder data path was proposed and proved TSC. However, ripple carry adders introduce a great delay so that they can be used for implementing only small adders. In order to implement great adders with acceptable delay, the proposed XOR gate and self-checking full adder can be adopted to implement fast adders (e.g., carry look-ahead, carry select, carry skip, etc.)

In order to make faster microprocessors, it is necessary to speed up the data path block, especially adders. Ripple carry adders exhibits the most compact design but the slowest in speed. Whereas carry look-ahead is the fastest one but consumes more area [9]. Carry select adders act as a compromise between the two adders [10].

Well developed schemes and proofs for SC carry look-ahead (full or group), carry-skip and conditional sum adders are presented in [11, 12].

In the following, we propose the use of the self-checking full adder and the differential XOR gate of figure 3 to build fast SC adders.

IV.1. The general structure of the self-checking data paths

The self-checking adder will be checked basically by the double-rail code. All the other blocks of the data path will be checked by the parity code. Using two different codes in the data path requires using one checker and one code translator for each adder block, and one checker and one code translator for each BUS. Using of translator circuits may involve an important hardware overhead. However, this does not occur in the present implementation. Indeed, a well known property of double-rail checkers is that they have a one-to-one correspondence with the parity trees. Therefore, the use of a translator to translate the double-rail code to the parity one is not necessary in the following implementations.

IV.2 carry look-ahead SC adder

Carry look-ahead (CLA) addition is one of the widely used methods in implementing fast adders [13], [14].
The carry look-ahead adder can be a group or a full one. In a group carry look-ahead adder each group of slices is implemented as a ripple carry adder. The carry-in signals for the groups are generated in a separate fast carry look-ahead unit. In the self-checking implementation these groups can be implemented using the double-rail code as previously.

The FA and differential XOR presented in the previous section and in [2] can be adopted for this scheme as shown in Fig. 4.
Fig. 4 gives the scheme of each slice of a full carry look-ahead adder, and the last slice of each group in the case of a group carry look-ahead adder.
All the carries generated by the carry look-ahead block (full or group), will be double-rail checked. The way this checking is given without duplicating the carry look-ahead block is presented in Fig. 4 and the proof is given in [11].

Each slice i receives two double-rail carry inputs $C_i, \overline{C_i}$ (generated by the previous slice i-1), and generates two double-rail carries $C_{i+1}, \overline{C_{i+1}}$. But the last slice of the group (e.g. the *slice k*) generates only the carry C_{k+1}. The carry look-ahead circuit generates also a second carry C_{k+1}. These two signals are compared by the carry double-rail checker as in Fig. 4.

Then, both the carry inputs $C_{k+1}, \overline{C_{k+1}}$ of the slice (*k+1)* (first slice of the next group) comes from the output C_{k+1} of the carry look-ahead circuit.
In the case of a full carry look-ahead each group of slices is reduced to a single slice as shown in Fig. 4.

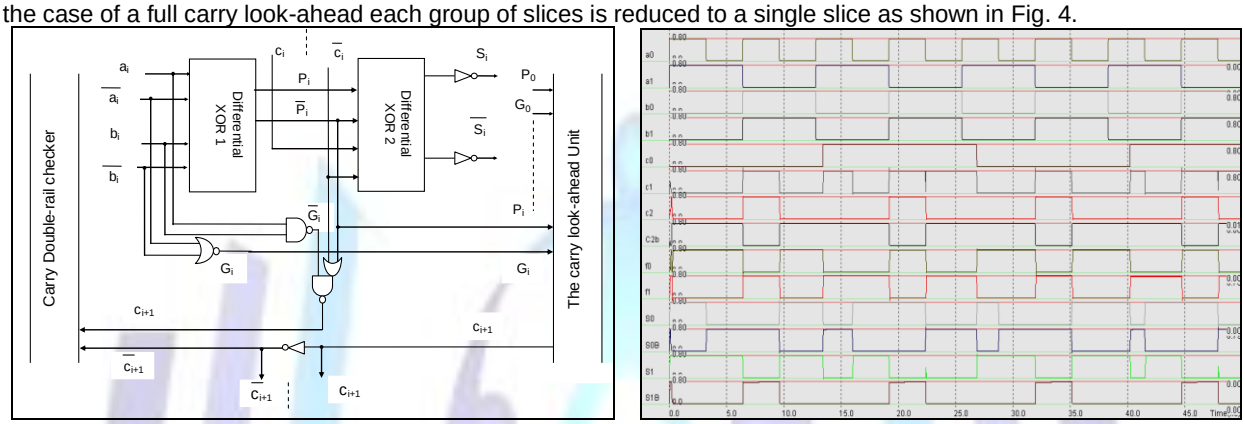

Fig 4: A full Carry look-ahead self-checking adder scheme and SPICE simulation

The outputs $S_i, \overline{S_i}$ are checked by a double-rail checker, this checker also generates the parity of the outputs. This checker is not shown in Fig. 4.
Table 1 gives the transistors count saving of the proposed CLA adder compared to duplication based SC Adder and the design presented in [11].

Table 1: Overhead saving (Tran. #)

Adder size (Bit #)	Duplication based SC CLA Adder	[6]
8	176	96
16	352	192
32	804	384
64	1608	768

IV. 2 Self-checking Carry skip SC adder

Another technique allowing fast carry propagation is the skip carry. In a carry-skip adder the carry signal skips a block if all the corresponding propagate signals of this block are equal to 1.

Since they compute the propagation signals, the proposed FA and differential XOR can easily, be adopted to implement SC carry skip adders.

A SC implementation carry skip adder based on Carry Checking/Parity Prediction technique is presented and analyzed in [11, 12].

The Fig. 5 illustrates how such an adder is implemented using the proposed differential XOR gate. We note that in this scheme, only a simple XOR (not differential) is needed to compute the FA outputs. Therefore, the differential FA of the Fig. 3 is modified and simplified as shown in Fig. 5.

ISSN 22773061

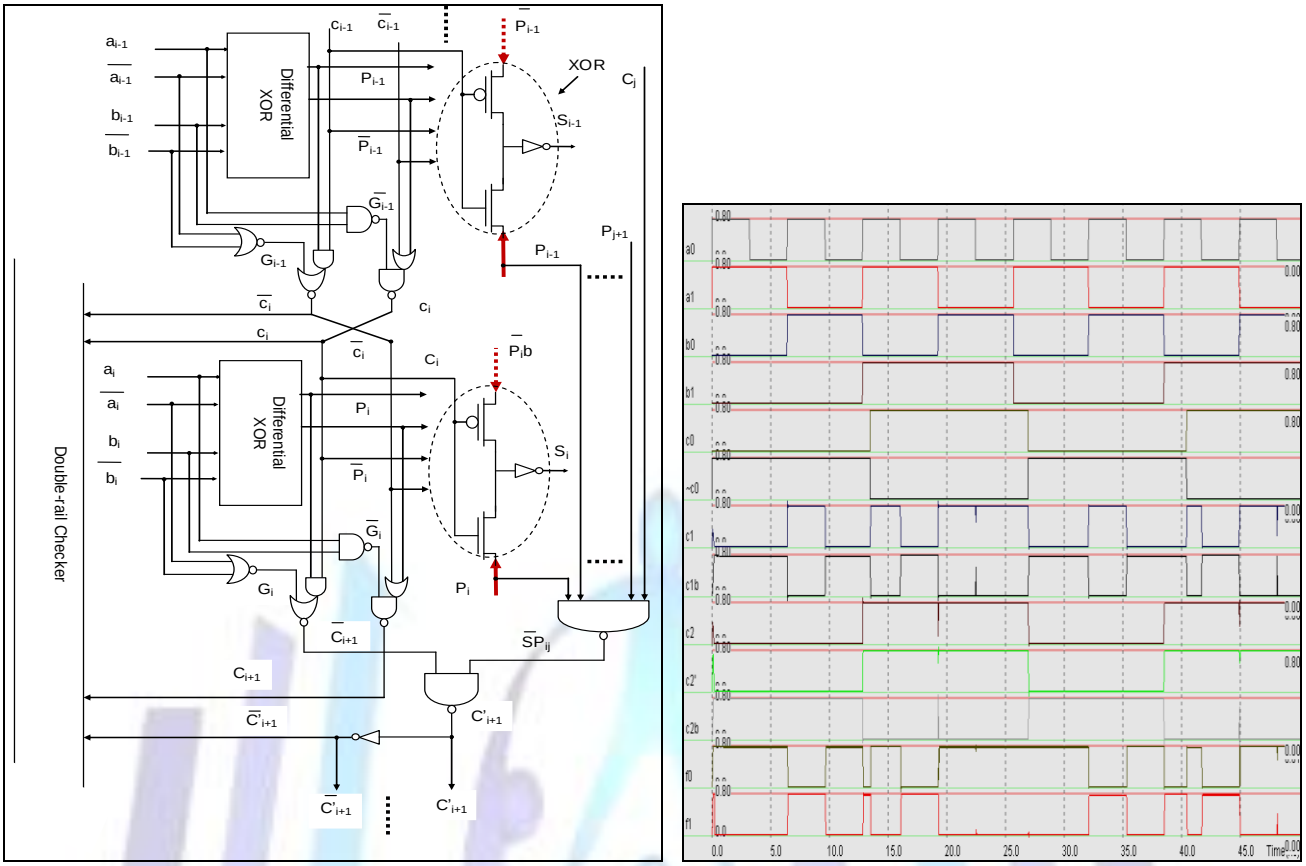

Fig 5: A Carry skip self-checking adder scheme and SPICE simulation (two bit Carry skip SC Adder)

The skip carry adders are implemented as the ripple-carry ones. However, in order to avoid delays due to long carry propagations, the output carry of some slices are merged with some signals SP_{ij} which propagate the carry from C_j to C_i (i.e. $SP_{ij} = C_j \,.\, P_{j+1}.\, P_{j+2} \ldots P_i$). The carry input C'_{i+1} of the slice (i+1) is generated from the carry output C_i of the slice (i) as following:

$$C'_{i+1} = \overline{\overline{SP_{ij}} \cdot \overline{C_{i+1}}} = SP_{ij} + C_{i+1}\,.$$

The signals C_{i+1} and C'_{i+1} compute the same function but C'_{i+1} is faster.
In the case of 16 bits SC carry skip adder, the implementation of the Fig. 5 saves 192 transistors, compared to the scheme of [11].

IV. 3 Self-checking Carry-select adder

The addition time of a ripple carry adder can be improved with a modified structure called the carry-select adder. The principle of a carry-select adder is to use one ripple carry adder to execute an addition assuming that the carry-in is "1". Another ripple carry adder is used to execute the same addition assuming that the carry-in is "0". Since the input carry signal of these blocks is always constant the hardware of theses blocks can be simplified.

The real carry-in computed in a previous stage is used to select one of the two sums with a multiplexor.

A SC carry-select adder is proposed in [15]. This schema is based on the ripple carry adder. It can be, therefore, easily implemented with the differential FA of the Fig. 3.

IV.4 Self-checking Conditional-Sum Adder

A conditional sum adder is an extension of carry select using one-bit blocks. It consists of conditional cells (Fig. 11(a)) and selection cells. The conditional cell is a modified full adder that computes the outputs S_i^0, S_i^1 and the carries out C_{i+1}^0, C_{i+1}^1.

S_i^0 and C_{i+1}^0 denote that the carry bit of the adder is 0, while S_i^1 and C_{i+1}^1 mean that the carry bit of the adder is 1. We have:

$$S_i^0 = a_i \oplus b_i = P_i$$

$$S_i^1 = a_i \oplus b_i \oplus 1 = \overline{P}_i \Rightarrow S_i^1 = \overline{S_i^0}$$

$$C_{i+1}^0 = a_i \cdot b_i = \overline{\overline{a}_i + \overline{b}_i}$$

$$C_{i+1}^1 = \ a_i + b_i \ + a_i.b_i = \ a_i + b_i \ = \overline{\overline{a}_i . \overline{b}_i}$$

Fig. 11(b) gives a two bits SC conditional sum adder implementation. This adder is fully controlled with double-rails checker as shown in Fig. 11(b). Fig. 11(c) gives SPICE simulation of a two bit SC conditional sum adder.

We could find S_i^0 and S_i^1 are complementary $S_i^1 = \overline{S_i^0}$, thus S_0^0, S_0^1, S_1^0 and S_1^1 could be connected to a first double-rail checker. We could also find that

$C_{i+1}^1 = (a_i \oplus b_i) + a_i.b_i = C_{i+1}^0 + S_i^0 \Rightarrow \overline{C_{i+1}^1} = \overline{C_{i+1}^0 + S_i^0}$. As shown in Fig. 11(b), C_1^1, $\overline{C_1^0 + S_0^0}$ and C_2^1, $\overline{C_2^0 + S_0^0}$ should be connected to a second double-rail checker.

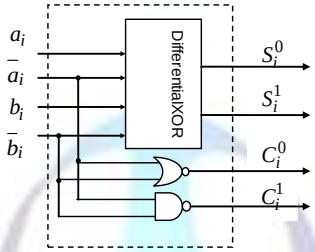

Fig 11(a): Conditional-sum cell

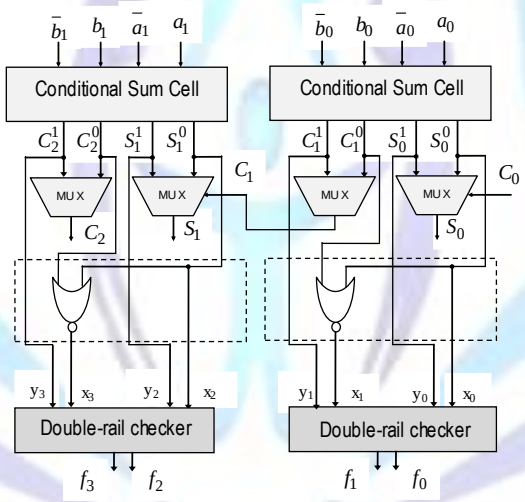

Figure 11(b): 2-bit self-checking conditional-sum adder module

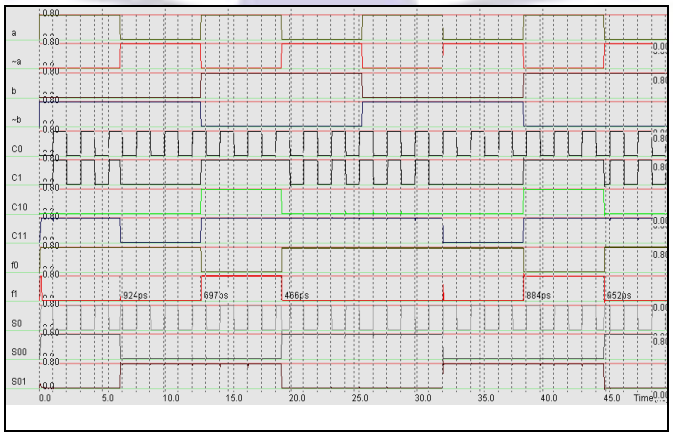

Figure 11(c): Two-bit self-checking conditional-sum adder SPICE simulation

Let's examine the behaviour of the circuit of Fig. 11(b) under any single stuck-at fault to make the proof that it is TSC for this class of faults.

In the *conditional-sum adder scheme of the Fig. 11(b),* faults can be classified into four categories:

- The fault happening to the differential XOR inputs or inside it
- The fault happening to the sum bit and carry bit of the adder
- The fault happening to the input or output of the NOR gate
- The fault happening inside the multiplexer

First category of faults:

In [1], we made the proof that the differential XOR gate is *totally self-checking* for all stuck-at, stuck-on and stuck-open single faults. Therefore, any non-code input word produce a non-code word, and will be transmitted to next stages.

Second category of faults:

When a stuck-at 0 (s@0) fault happens to the sum bit S^1_0, the wrong value 0 will be transmitted to y_0, and will not be complementary to the value transmitted to x_0. The fault inside the adder could then be affirmed accordingly by the double-rail checker. When a stuck-at 1(s@1) fault happens to the carry bit C^0_1, the wrong value 1 will be transmitted to x_1 and will not be complementary to the value transmitted to y_1. The fault inside the adder could then also be detected accordingly. If faults happen to S^1_0 and C^0_1 at the same time, the faults inside the adder could be identified from the output signals of the two-rail checker.

third category of faults:

When a fault happens to the input or output of the NOR gate and a wrong value as the input is transmitted to the two-rail checker, the outputs f_0 and f_1 of the two-rail checker will indicate non-complementary signals (non-code word). The fault inside the adder could then be detected accordingly. The SC propriety proof for the fault falling into this category is the same as that of the stuck-at-fault at C^0_1 in the second category.

Fourth category of faults:

We have two cases: fault happening inside the first multiplexer which generate the output S_i. This will be checked by the parity code for the whole data path scheme as it the case for SC ripple carry adder data path [2]. the fault happening inside second multiplexer which generate the carry signal C_i, will be propagated to next stage, and therefore detected by the same technique (parity checking).

From the above discussion, the proposed schemes for self-checking fast adders data paths can be said optimal [8] since it had reached the following goals:

- They are self-checking for all the single faults,
- They requires a low hardware overhead,
- They are checked by a compact checker,
- They can be combined with a parity checked data path without using code translators.

VI. CONCLUSION

In this paper, we proposed novel schemes of dual-rail based self-checking fast adder's data paths. We combined Conventional CMOS and CPL CMOS styles to achieve low overhead and low power design. We presented an efficient Self-checking schemes for carry look-ahead, carry skip, Carry-select and Conditional-Sum adder data paths. These implementations are based on a TSC differential full adder and a CPL TSC differential XOR gate. SPICE simulations of the proposed schemes, including parasitic, are performed to demonstrate that these design had an acceptable electrical behaviour.

REFERENCES

[1]. Hamdi Belgacem, Khedhiri Chiraz, and Tourki Rached, "A novel differential XOR-based selfchecking adder". International Journal of Electronics, 2012, 1–23, iFirst, Taylor & Francis.

[2]. Belgcem Hamdi , Chiraz Khedhiri , Aymen Fradi and Tourki Rached, "Four Transistors Self-checking Differential XOR". In Proceedings of the 1^{0th} Edition IEEE International Symposium on Signals, Circuits & Systems, Romania, June, 2011.

[3]. Michael Nicolaidis, On-line testing for VLSI: state of the art and trends, Integration, the VLSI Journal, Volume 26, Issues 1-2, 1 December 1998, pp. 197-209.

[4]. D. K. Pradhan, J. J. Stiffler, Error correcting codes and self-checking circuits in fault-tolerant computers. IEEE Computer Mag. Vol. 13, March 1980, pp. 27-37.

[5]. A.P. Chandrakasan, S. Sheng and R. W. Brodersen, Low-Power CMOS Digital Design. IEEE Journal of Solid State Circuits, Vol. 27, No. 4, April 1992, pp. 473–484.

[6]. D. A. Anderson and G. Metze, Design of totally self-checking check circuits for m-out-of-n codes. IEEE Trans. on Computers, vol. 22, No. 3, March 1973, pp. 263-269.

[7]. P. Oikonomakos, M. Zwolinski, On the Design of Self-Checking Controllers with Datapath Interactions. IEEE Transactions on Computers, Volume 55, No 11, Nov 2006, pp. 1423 – 1434.

[8]. M. Nicolaidis, L. Anghel, Concurrent Checking in VLSI. Microelectronic Engineering, Vol. 49, Nov 1999, pp 139-156.

[9]. A.Tyagi, "A reduced area scheme for carry-select adders", IEEE Trans. on Computer, vol. 42, pp. 1163- 1170, 1993

[10]. Padma Devi, Ashima Girdher and Balwinder Singh, "Improved Carry Select Adder with Reduced Area and Low Power Consumption, "International Journal of Computer Applications (0975 – 8887), Volume 3 – No.4, June 2010.

[11]. M. Nicolaidis, Efficient implementations of self-checking adders and ALUs, in: Proceeding of 23rd International Symposium on Fault-Tolerant Computing, June 1993, pp. 586-595.

[12]. B. Hamdi, H. Bederr, M. Nicolaidis, A tool for automatic generation of self-checking data paths, in: Proceeding of 13th IEEE VLSI Test Symposium, Proceedings (VTS'95), 30 Apr-3 May 1995, pp 460 – 466.

[13]. A.Naini et al., "A 4.5 ns 96 b adder design," in CICC Proc. pp. 25.5.1–25.5.4, 1992.

[14]. Y. Tsujihashi et al., "A high density data path generator with stretchable cells," in CICC Proc. pp. 11.3.1–11.3.4, 1992.

[15]. B. K. Kumar and P. K. Lala, On-Line Detection of Faults in Carry-Select Adders, in Proc. Int. Test Conf., vol. 1, Charlotte, NC, Sep. 30-Oct. 2, 2003, pp. 912–918

Authors:

Belgacem HAMDI is with the Electronic & microelectronic's LAB, Monastir, Tunisia. PH.D. in Microelectronics from INP Grenoble (France). Assistant Professor at ISSAT Sousse, Tunisia. His main areas of interest are: IC design, Test, Built In self Test, DFT tools, self-checking and Fault tolerant systems.

E-mail: belgacem.hamdi@issatgb.rnu.tn; belgacem.hamdi@gmail.com

Khaled Ben Khalifa He is actually an assistant professor on Electrical Engineering. He received his philosophy doctor degree in Physics from the Faculty of Sciences of Monastir in 2006. His research interests are VLSI design, Implementation and Optimization of Artificial Neural Networks and Embedded Systems.

E-mail: khaled.benkhalifa@issatso.rnu.tn

Aymen FRADI He completed his Masters in Electronics and actually is a PHD student, attached to the Electronic & microelectronic's LAB, at the University of Monastir in Tunisia

E-mail: aymenfradi@gmail.com