Designing FPGA based Self-Testing Checkers for m-out-of-n Codes

A. Matrosova
Tomsk State University, Russia
mau@fpmk.tsu.ru

V. Ostrovsky
Tel Aviv University, Israel
vios@post.tau.ac.il

I. Levin
Tel Aviv University, Israel
i.levin@ieee.org

K. Nikitin
Tomsk State University, Russia
nikitin@ fpmk.tsu.ru

Abstract

The paper describes a specific method for designing self-checking checkers for m-out-of-n codes. The method is oriented to the Field Programmable Gate Arrays technology and is based on decomposing the sum-of-minterms corresponding to an m-out-of-n code. The self-testing property of the proposed checker is proven for a set of multiple stuck-at faults at input and output poles of a Logic Cell. An estimated complexity of obtained m-out-of-n checker demonstrates high efficiency of the proposed method.

1. Introduction

This paper considers the problem of designing self-testing *m-out-of-n* code checkers on the base of Field Programmable Gate Array (FPGA). In most of the works related to the field, methods of designing self-checking schemes are based on the two following principal assumptions associated with fault models [1]:

- Only one fault may occur in the scheme at a time. In other words, before a next fault occurs, the previous fault will be eliminated.
- A distortion caused by a fault can only be unidirectional, i.e. only low or only high positions of output codewords switch their values to the opposite due to the fault.

It is known, however, that the above assumptions are non-applicable for some types of the schemes. For example, CMOS-based schemes are known as such where not a single element, but an area forming a number of elements might be damaged. Another example is a LUT-based FPGA scheme, where the assumption of unidirectional distortion of output codewords is incorrect.

In light of the above, the fault model requires to be changed. The present paper proposes a new model for describing faults in LUT-based FPGAs. According to the proposed model, we assume that any number of signals at inputs or outputs of a single logic cell (LC) may be distorted simultaneously. We consider only constant distortions and assume that both input and output poles cannot be stuck-at simultaneously. We will call a set of those distortions **F**-set. We assume that a fault is detected on a set of input codewords, if that set comprises at least one input codeword, which leads to an output codeword differing from any of the correct ones. LC may have one output or two outputs.

In the paper, the above fault model is assumed in the proposed method for synthesis self-checking checkers of *m-out-of-n* code. The self-testing checker design, as a rule, is based on determining the weight of input codewords of a checker. For this aim either threshold circuits [2-7] or circuits based on parallel counters [8-11] can be applied. All these checkers are oriented to gate implementation.

In this paper we propose a universal decomposition method to design any *m-out-of-n* codes checkers by FPGAs. It is proven that the checker is self-testing for the set **F** of faults that covers considerably more realistic failures of a circuit in comparison with traditional single stuck-at faults.

Section 2 of the paper describes a decomposition method for a self-testing *m-out-of-n* codes checker. Section 3 discusses design of a self-testing *m-out-of-n* codes checker. The self-testing property of the proposed checker is discussed in Section 4. Evaluation of results presented in Section 5.

2. Decomposition Formula

Let us put a certain minterm into correspondence to each of *m-out-of-n* codeword. Denote sum of such minterms (SOM) as $D_n^m(X)$. For example, $D_{10}^5(X)$ consists of 252 minterms of rank 10 and contains 2520 literals. This expression cannot be minimized since any two minterms of $D_n^m(X)$ are at least bi-directional [12].

For a compact description of all *m-out-of-n* codewords, a specific formula (1) is proposed below.

$$D_n^m(X) = \vee D_k^i(X^1) D_{n-k}^j(X^*) \qquad (1)$$

Here symbol $\wedge$ between $D_k^i(X^1)$, and $D_{n-k}^j(X^*)$ is omitted. Set X is divided into two subsets: $X^1 = \{x_1, ..., x_k\}$ and $X^* = \{x_{k+1}, ..., x_n\}$.

Call $D_{n-k}^j(X^*)$ as a decomposition coefficient. If $n-k>k$, then execute the next step of the decomposition (1) for each $D_{n-k}^j(X^*)$, and so on. As a result, we obtain the formula (1) in which for any $D_p^q(X^r)$ the condition $p \le k$ takes place.

Consider an example. Obtain decomposition formula for D_6^3, $k=2$.

First $X^1 = \{x_1, x_2\}$, $X^* = \{x_3, x_4, x_5, x_6\}$,

$$D_6^3 = D_2^0(X^1)D_4^3(X^*) \vee D_2^1(X^1)D_4^2(X^*) \vee D_2^2(X^1)D_4^1(X^*).$$

Then $X^2 = \{x_3, x_4\}$, $X^3 = \{x_5, x_6\}$. Execute the next step of decomposition (1) for $D_4^3(X^*), D_4^2(X^*), D_4^1(X^*)$. We arrive at the following:

Proceedings of the 9th IEEE International On-Line Testing Symposium (IOLTS'03)
0-7695-1968-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

$$D_6^3 = D_2^0(X^1)(D_2^1(X^2)D_2^2(X^3) \vee$$
$$\vee D_2^2(X^2)D_2^1(X^3)) \vee D_2^1(X^1)(D_2^0(X^2)D_2^2(X^3) \vee$$
$$\vee D_2^1(X^2)D_2^1(X^3) \vee D_2^2(X^2)D_2^0(X^3)) \vee$$
$$\vee D_2^2(X^1)(D_2^0(X^2)D_2^1(X^3) \vee D_2^1(X^2)D_2^0(X^3)).$$

Let ν be the number of subsets $X^1,\ldots,X^\nu$ on which a set X of variables of $D_n^m(X)$ is divided with deriving the formula (1), $\left|X^1\right| = \ldots = \left|X^{\nu-1}\right| = k$, $\left|X^\nu\right| \le k$ and (ν-1) is the number of the decomposition steps. In our example, $\nu = 3$ and we have two decomposition steps.

Architecture of the checker corresponding to formula (1) is discussed below in Section 3.

3. Implementation of self-testing checker

Divide whole set of *m-out-of-n* codewords A into two subsets A^1andA^2 ($A = A^1 \vee A^2$). For example, A^1 represents the codewords resulted from the product $D_2^0(X^1)D_4^3(X^*)$ and A^2 – the codewords resulted from two products: $D_2^1(X^1)D_4^2(X^*)$, $D_2^2(X^1)D_4^1(X^*)$.

Let's obtain a two-output self-testing circuit C (*m-out-of-n* code checker) that implements A^1 and A^2 formulae on its outputs. Since each couple of codewords is orthogonal, an output vector of the scheme C can be equal either 10 or 01. Let different variables $y_1, y_2, \cdots, y_{2l-1}, y_{2l}$ correspond to different decomposition functions and decomposition coefficients of the expression (1).

We have:

$$D_n^m(x) = y_1y_2 \vee y_3y_4 \vee \cdots \vee y_{2l-1}y_{2l} \quad . \qquad (2)$$

Now cover expression (2) with logic cells (LC). Any LC has k inputs. Let k be an even number, $k = 4$. We assume here that the number of products $l > 1$. If the number of products in the expression (2) is less or equal to $0.5k$, then is realized by one LC. If $l > 0.5k$, execute the following steps.

1. Separate the first $0.5k$ minterms of expression (2). Let LC_1 implements these codewords. For providing the self-testing property, transform all products of expression (2) to a form that includes all input variables of LC_1. For that purpose, include into the expression all missing variables in their negative form. Such a transformation is acceptable since any *m-out-of-n* codeword does not activate more then one product of expression (2). For example, let we have an expression $y_1y_2 \vee y_3y_4 \vee y_5y_6$. If $k = 4$, then the expression $y_1y_2 \vee y_3y_4$ has to be replaced by $y_1y_2\bar{y}_3\bar{y}_4 \vee \bar{y}_1\bar{y}_2y_3y_4$. Such transformations will be performed for all functions implemented on the logic cell outputs.

2. Three cases can be considered:

a) The number of remaining products of expression (2) is equal to $0.5k + 1$. Then we use one output LC_2 to implement the next $0.5k$ products from (2). Assign variables z_1, z_2 to outputs of the LC_1, LC_2, correspondingly. Logic cells LC_3 and LC_4 implement the following products (Table 1).

Table 1. Products implemented by LC_3 and LC_4 for *l=k+1*

z_1	z_2	y_{2l-1}	y_{2l}
1	0	0	0
0	1	0	0
0	0	1	1

This table lists all products on which the LC function tales value "1". All products are presented as Boolean vectors. LC_3, LC_4 implement two functions corresponding to outputs of the circuit C. One of these functions is presented by the two first lines (for LC_3) of Table 1, the second function – by the last line (for LC_4). A subcircuit of C implementing such expressions is shown in Fig. 1.

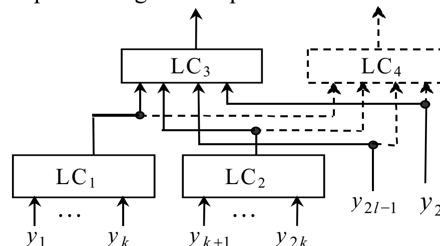

Figure 1. Implementation of (2) for *l=k+1*

b) The number of remaining products of the expression (2) is equal to $0.5k$. We use LC_2 to implement sum of these products. Table 2 represents the following products implemented by LC_3, LC_4 similar to Table 1. Corresponding subcircuit of C is shown in Fig 2.

Table 2. Products implemented by LC3 and LC4 for l=k

Z_1	Z_2
1	0
0	1

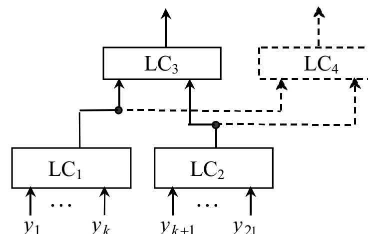

Figure 2. Implementation of (2) for l=k

c) The number of remaining products of expression (2) is less then $0.5k$. Table 3 and Figure 3 illustrate this case.

0-7695-1968-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

Table 3. Products implemented by LC_2 and LC_3 for l<k

z_1	$y_{(k+1)}$	$y_{(k+2)}$	$\dots$	$y_{(2l-1)}$	$y_{(2l)}$
1	0	0	$0\dots0$	0	0
0	1	1	$0\dots0$	0	0
			$\vdots$		
0	0	0	$0\dots0$	1	1

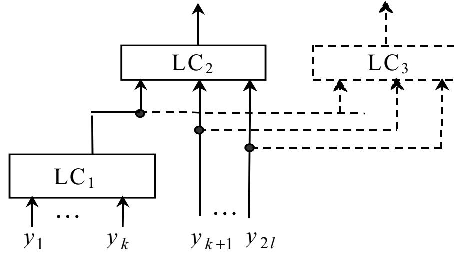

Figure 3. Implementation of (2) for l<k

Thus, we need no more than four LCs to implement expression (2).

Notice, that:

- any product of the LC function is activated at least once during occurrence of all *m-out-of-n* codewords.
- any product of the decomposition function which implements LC is activated at least once during arriving all *m-out-of-n* codewords.

A subcircuit implementing decomposition functions of the i-th level is shown in Fig. 4. The subcircuit has k (or less) inputs and l_i outputs, which correspond to the different decomposition functions at the i-th level. The total number of LCs is this implementation is equal to $\sum_i 0.5 l_i$.

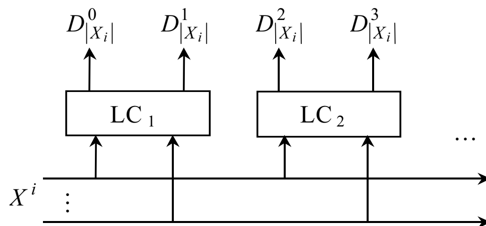

Figure 4. Implementation of i-th level decomposition functions

To design the two-outputs *m-out-of-n* -checker C by LCs, we first need to implement all decomposition functions. Then we have to implement all expressions (2) corresponding to (1) and, finally, to combine all of them into checker C.

Let us consider an example of a self-testing checker design using 4-input LCs. The checker is *6-out-of-12* code checker.

Let us partition $X = \{x_1, \dots x_{12}\}$ into 3 subsets:

$X^1 = \{x_1, \dots, x_4\}, X^2 = \{x_5, \dots, x_8\},$

$X^3 = \{x_9, \dots, x_{12}\}$

We have the following decomposition expressions:

$$D_{12}^6(X) = D_4^0(X^1) D_8^6(X^2, X^3) \vee$$
$$\vee D_4^1(X^1) D_8^5(X^2, X^3) \vee D_4^2(X^1) D_8^4(X^2, X^3) \vee$$
$$\vee D_4^3(X^1) D_8^3(X^2, X^3) \vee D_4^4(X^1) D_8^2(X^2, X^3),$$

$$D_8^2(X^2, X^3) = D_4^0(X^2) D_4^2(X^3) \vee$$
$$\vee D_4^1(X^2) D_4^1(X^3) \vee D_4^2(X^2) D_4^0(X^3),$$

$$D_8^3(X^2, X^3) = D_4^0(X^2) D_4^3(X^3) \vee$$
$$\vee D_4^1(X^2) D_4^2(X^3) \vee D_4^2(X^2) D_4^1(X^3) \vee$$
$$\vee D_4^3(X^2) D_4^0(X^3),$$

$$D_8^4(X^2, X^3) = D_4^0(X^2) D_4^4(X^3) \vee$$
$$\vee D_4^1(X^2) D_4^3(X^3) \vee D_4^2(X^2) D_4^2(X^3) \vee$$
$$\vee D_4^3(X^2) D_4^1(X^3) \vee$$
$$\vee D_4^4(X^2) D_4^0(X^3),$$

$$D_8^5(X^2, X^3) = D_4^1(X^2) D_4^4(X^3) \vee$$
$$\vee D_4^2(X^2) D_4^3(X^3) \vee D_4^3(X^2) D_4^2(X^3) \vee$$
$$\vee D_4^4(X^2) D_4^1(X^3),$$

$$D_8^6(X^2, X^3) = D_4^2(X^2) D_4^4(X^3) \vee$$
$$\vee D_4^3(X^2) D_4^3(X^3) \vee D_4^4(X^2) D_4^2(X^3).$$

$D_4^0(X^1), D_4^1(X^1), D_4^2(X^1), D_4^3(X^1), D_4^4(X^1)$ – the decomposition functions of the first level. Their implementation demands 2 two-output LCs and 1 one-output LC. Thus, the subcircuit that realizes these functions consists of 3 LCs.

$D_4^0(X^2), D_4^1(X^2), D_4^2(X^2), D_4^3(X^2), D_4^4(X^2)$ – the decomposition functions of the second level. The subcircuit that realizes these functions is similar to that mentioned above.

$D_4^0(X^3), D_4^1(X^3), D_4^2(X^3), D_4^3(X^3), D_4^4(X^3)$ – the decomposition functions of the third level. They all are implemented with a subcircuit similar to those mentioned above. We need 9 LCs to implement all decomposition functions of the checker considered.

To implement expression (2) for $D_8^2(X^2, X^3)$ we need 2 LCs, $D_8^3(X^2, X^3)$ – 3 LCs, $D_8^4(X^2, X^3)$ – 3 LCs, $D_8^5(X^2, X^3)$ – 3 LCs, at least $D_8^6(X^2, X^3)$ – 2 LCs.

0-7695-1968-7/03 $17.00 © 2003 IEEE
IEEE COMPUTER SOCIETY

Moreover, we need 4 LCs to implement the expression (2) for $D_{12}^{6}(X)$.

It is enough to have 17 LCs to realize all expressions (2) of the self-testing checker. Consequently, for the checkers as a whole, we need 26 LCs.

4. Self-testing property of the checker

Recall that we deal with a set **F** consisting of multiple stuck-at faults at the LC input poles and multiple stuck-at faults at the LC output poles for each LC. We use both one output and two outputs LCs. Input and output poles of the same LC cannot have stuck-at faults simultaneously.

Any LC implementing formula (1) represents each its function *f* as a sum of minterms. Hereafter we will identify the minterms and representing them Boolean vectors.

We will describe multiple stuck-at faults on input poles by a ternary vector β as follows. If stuck-at input has the 1(0) value then the corresponding component of β takes also the 1(0) value. Call all such components as determined ones. The remaining components of β take a "don't care" value. Call them as undetermined components of β. If undetermined components are absent, vector β turns into a Boolean vector.

Consider a certain LC function *f* and the corresponding output. Let's say that we deliver different Boolean vectors of the length k to the LC inputs, and observe the associated LC output. Now we investigate the condition on which the Boolean vector is a test pattern for the fault represented with β. β divides all minterms of *f* into two subsets $M_{\beta}(f)$ and $M_0(f)$. Subset $M_{\beta}(f)$ includes minterms (Boolean vectors) absorbed with β, and the subset $M_0(f)$ - minterms that are orthogonal to β. Having excluded determinate variables of β from the Boolean vectors of $M_{\beta}(f)$ and $M_0(f)$, we obtain the set $M'_{\beta}(f)$, $M'_0(f)$, respectively.

For example, let us LC implements a function D_4^2 represented in Table 4 and let us vector β is equal to (- 1 - 0). It means that the input represented as x_2 is stuck-at 1 and the input represented as x_4 is stuck-at 0. The rest of the LC inputs x_1, x_3 are fault free.

Table 4. *2-our-of-4* codewords

0	0	1	1
0	1	0	1
1	0	0	1
0	1	1	0
1	0	1	0
1	1	0	0

Then: $M_{\beta}\left(D_4^2\right) = \{(0110), (1100)\}$, $M_0\left(D_4^2\right) = \{(0011), (0101), (1001), (1010)\}$.

After excluding determined variables from β, we obtain $M'_{\beta}\left(D_4^2\right) = \{(01), (10)\}$ and $M'_0\left(D_4^2\right) = \{(01), (00), (10), (11)\}$.

Call the number of the 1-value components of β its weight, represented by λ. In our example: $\lambda = 1$.

Take into account that a Boolean vector $\alpha \in M_0\{\beta\}$ delivered to the LC inputs is changed by the fault for the Boolean vector $\beta * \alpha', \alpha' \in M'_0(f)$ where α' resulted from the corresponding α. Here $\beta * \alpha'$ means the determined components of β are added to the components of α'. If $\alpha \in M_{\beta}(f)$ then α is not changed by the fault.

In the above-mentioned example for $\alpha = (0011), \alpha \in M_0(D_4^2)$, we have $\alpha' = (01)$ and $\beta * \alpha' = (0110)$, for $\alpha = (0101), \alpha \in M_0(D_4^2)$, we have $\alpha' = (00)$ and $\beta * \alpha' = (0100)$.

Theorem 1. If α' from $M'_0(f)$ is orthogonal to each vector from $M'_{\beta}(f)$ then the Boolean vector $\alpha \in M_0(f)$ corresponding to α' is a test pattern for the fault represented by β.[1]

In the example the vector $\alpha = (0101)$ results in $\alpha' = (00)$, and α' is orthogonal to vectors 01, 10 from $M'_{\beta}\left(D_4^2\right)$, $\beta * \alpha' = (0100)$. The latter vector is missed in the expression $M_{\beta}\left(D_4^2\right) \cup M_0\left(D_4^2\right)$. Consequently, $\alpha = (0101)$ is the test pattern for the fault.

Let a certain LC function f be represented with $D_p^q, 0 < q < p$. Notice that each of the variables of function f takes both 1 and 0 values for all products (Boolean vectors) of D_p^q, which means that *f* is not unate on each variable.

Theorem 2. A test pattern exists for any multiple stuck-at fault at the LC input poles for a LC implementing two decomposition functions $D_p^{q_1}$ and $D_p^{q_2}$ so that $0 < q_1 < p$. This test pattern is a Boolean vector, representing one product either from $D_p^{q_1}$ or $D_p^{q_2}$. We assume that both LC outputs are observable and they are not outputs of a checker.

Corollary 2.1. Using two outputs LC to implement decomposition functions we may combine $D_p^{q_2}$ when $q_2 = 0$ or $q_2 = p$ with $D_p^{q_1}, 0 < q_1 < p$.

[1] We don't provide proofs of the theorems in this paper, due to the space limitations

IEEE COMPUTER SOCIETY

Corollary 2.2. When a multiple stuck-at fault at LC input poles occurs and input vector α does not coincide with any product of $D_p^{q_1}$ and $D_p^{q_2}$, it is possible that the values 10, 01, 11 will appear instead of 00 at the LC outputs (1 instead of 0 for one output LC).

Corollary 2.3. If Checker C is implemented with two one output CLs any fault from a set F is detectable.

For one output LC comprises a subcircuit implementing the expression (2), we will have the following.

1. For each input variable, only one minterm of the LC function exists for which this variable takes value 1.
2. No input variable is unate for minterm of the LC function.
3. LC input variables are divided into two subsets Y and Z. One of these can be empty. Any minterm of the LC function has 1-value components only among one of these subsets.

Theorem 3. For one output LC that comprises a subcircuit implementing the expression (2), ν>1, either there exists a test pattern for multiple stuck-at fault at the LC input poles or this fault manifests itself as single stuck-at fault at this LC output.

Corollary 3.1. When a multiple stuck-at fault at the LC input poles occurs, and an input vector α does not coincide with any products of the LC function it is possible 1 instead of 0 at the LC output

Corollary 3.2. For one output LCs that comprises a subcircuit implementing the expression (2), *ν=1*, so that their outputs are at the same time outputs of a checker any fault from **F** is detectable.

Theorem 4. A checker *C* is self-testing for a set **F** of faults.

This Theorem follows directly from Theorems 1, 2, 3.

5. Evaluation of results

Estimations of completeness of the checker designed according the proposed method are presented in Table 5.

Table 5. Estimation of checker's completeness

	D_5^2	D_6^2	D_7^2	D_9^2	D_6^3	D_8^4	D_{10}^5	D_{12}^6
LC	3	7	7	9	7	10	19	26
LC	8	11	-	-	6	15	25	37

The first row of Table 5 illustrates the numbers of LCs we need for certain checkers when applying the decomposition method. The second row of this table illustrates the numbers of LCs we need after covering by LCs the best gate based checker implementations [6]. The columns of Table 5 correspond to the different checkers marked by D_n^m. As can be seen from Table 5, the proposed method provides decreasing of the checker's complexity for about 30% in comparison with [6, 12].

6. Conclusions

In this paper, we presented a new decomposition method for designing *m-out-of-n* self-testing checkers. The method is suitable for implementation of the checkers by FPGA. The self-testing property is provided for a wide set of faults, namely multiple stuck-at faults at the LC input and output poles.

Comparison of overheads performed for the *m-out-of-n* checkers implemented according to the proposed method and *m-out-of-n* checkers designed using known methods for synthesis, shows that the proposed approach leads to a considerable overhead reduction in most cases. It is important to note that the most significant overhead reduction can be achieved for large values of *m* and *n*.

A new fault model was proposed. This fault model allows describing faults leading to both unidirectional or arbitrarily errors.

References

[1] M. Nicolaidis, Y. Zorian. "On-Line Testing for VLSI—A Compendium of Approaches", *Journal of Electronic Testing: Theory and Applications* 12, 7–20 (1998), Kluwer Academic Publishers, 1998.

[2] D.A. Anderson and G. Metze, "Design of Totally Self-Checking Circuits for m-out–of-n Codes," *IEEE Trans. Computer*, Vol. C-22, pp. 263-269, March 1973.

[3] S.J. Piestrak, "The Minimal Test Set for Sorting Networks and the Use of Sorting Networks in Self-Testing Checkers for Unordered Codes," *Dig. Pap. FTCS-20*, Newcastle upon Tyne, UK, June 1990, pp. 457-464.

[4] S. J. Piestrak, "Design Method of Totally Self-Checking Checkers for m-out-of-n Codes", *Dig. Pap. FTCS-13*, Milan, Italy, pp. 162-168, June1983.

[5] S. J. Piestrak, Design of Fast Self-Testing Checkers for m-out–of-2m and m-out-of-$(2m \pm 1)$ Codes", *Int. J. Electronics*, Vol. 74, pp. 177-199, Feb. 1993.

[6] S. J. Piestrak, "Design of Self-Testing Checkers for Unidirectional Error Detecting Codes", *Scientific Papers of Inst. of Techn. Cybern. of Techn. Univ. of Wroclaw*, No. 92, Ser.: Monographs No. 24, Wroclaw, 1995.

[7] S. J. Piestrak, "Design of Encoders and Self-Testing Checkers for Some Systematic Unidirectional Error Detecting Codes", *Proceedings of the 1997 Workshop on Defect and Fault-Tolerance in VLSI Systems (DFT '97).*

[8] V.V. Dimakopoulos et al., "On TSC Checkers for m-out-of-n Codes," *IEEE Trans. Comput.*, Vol. 44 pp. 1055-1059, Aug.1995.

[9] C. Efstathiou and C. Halatsis, "Efficient Modular Design of m-out–of-2m TSC Checkers, for m=2^K-1, K>2", *Electron. Lett.*, Vol. 21, pp. 1082-1084, Nov. 1985.

[10] A.M. Paschalis, "Efficient Structured Design of Totally Self-Checking M-out-of-N Code Checkers with N>2M and M=2^K-1", *Int. J. Electronics*, Vol. 77, pp. 251-257, Aug. 1994.

[11] A.M. Paschalis, D. Nicolos, and C. Halatsis, "Efficient Modular Design of TSC Checkers for m-out-of-n Codes", *IEEE Trans. Comput*, Vol. C-37, pp. 301-309, March 1988.

[12] P. Lala. *Self-Checking and Fault Tolerant Digital Design*. Morgan Kaufman Publishers, 2001.

0-7695-1968-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY