Approach to partially self-checking combinational circuits design

Goran Lj. Djordjevic[1], Mile K. Stojcev, Tatjana R. Stankovic

Faculty of Electronic Engineering, University of Nis,
Beogradska 14, 18000 Nis, Serbia & Montenegro

Abstract

This paper presents a cost-effective, non-intrusive technique of partially self-checking combinational circuits design. The proposed technique is similar to duplication with comparison, wherein duplicated function module and comparator act as a function checker that detects any erroneous response of the original function module. However, instead of realizing checker with full error-detection capability, we select a subset of erroneous responses to implement partial, but simplified function checker. A heuristic procedure that tries to find the optimal sum-of-product expression for partial function checker that minimizes its area while providing specified error coverage is described here. Effectiveness of the technique is evaluated on a set of MCNC 91 benchmark combinational circuits.

Keywords: fault tolerance, concurrent error detection, partially self-checking circuits, approximation of logic functions.

1. Introduction

[1] **Corresponding author**: G.Lj.Djordjevic: E-mail: gdjordj@elfak.ni.ac.yu; Tel.: +381-18-547597; fax: +381-18-524931;

Advances in semiconductor technology enable much more complex applications on the integrated circuits. The main trends are oriented now towards process technology scaling (<100 nm), increased levels of integration densities, higher performance and reliability, greater functionality and speeds, lower voltage operation, low-power and power-aware consumption, lower cost, etc. [1]. Studies indicate that circuits will become increasingly sensitive to transient and intermittent faults caused by crosstalk, supply and power supply noise, charge sharing, terrestrial cosmic rays and alpha particles, smaller noise margins, etc., and this will result in unacceptable soft error rates in logic circuits [2].

Circuit-level techniques for concurrent error detection (CED) permit early detection and containment of errors before they can propagate to other parts of the system and corrupt data [3]. The basic objectives behind the use of CED are: (1) detection of errors as early as possible so that corrective action can be initiated before data corruption; and (2) identifying the field replicable unit [4]. Previous research in CED is focused on mission critical systems where very high levels of dependability are required. In these applications, the main goal is to provide high levels of reliability, while the cost of error detection circuitry is of less importance. Today (with channel size <100nm), as CED becomes increasingly important in commercial electronics, there is a need for more cost-effective techniques. In addition, it is widely accepted that the fault coverage for permanent fault is less than 100 % [5].

Typically, computed results are verified by using a self-checking design technique, primarily because the self-checking property allows both transient/intermittent and permanent faults to be detected, thus preventing data contamination. Self-checking circuits consist of a functional unit encoded by means of an error detecting code and are continuously verified by the checker. Existing general approaches for designing self-checking circuits are based on duplication [7,8] or application of specific error detecting codes (Berger code [4, 9], Bose-Lin code [10], *m*-out-of-*n* codes [11], constant-weight codes [12], parity code [6,13], etc.) each of them with its own characteristics, regarding error detection capabilities, hardware overhead, and checker complexity. In most cases, especially for arbitrary logic function, these approaches require a hardware overhead of more than 100 % [6, 8, 14]. In general, the optimal synthesis of self-checking circuits for any kind of code is still an open problem.

Duplication-related self-checking techniques are of particular interest for our work. In general, duplication in the form of self-checking pairs is the simplest form of redundancy that can be used for CED. This comparison-based scheme imposes an identity property between the original output and the replica output, which may be simply examined by a comparator (see Fig. 1(*a*)). With the exception of common-mode failures, duplication will immediately detects all errors, but incurs significant hardware overhead that exceeds 100% of the cost of the original circuit [15].

Several duplication-like design solutions for achieving soft-error tolerance were proposed in [2, 18]. The partial duplication approach described in [2] exploits the asymmetric soft-error error susceptibility of nodes in a logic circuit to achieve cost-effective tradeoffs between overhead and reduction in the soft error failure rate. This technique first characterizes the soft error susceptibility of nodes in logic circuit, and then selects the set of nodes with highest soft error susceptibility for partial duplication. However, this technique is intrusive and limited to multilevel logic networks. Reference [18] considers a scheme that exploits the temporal nature of soft-errors in order to detect them by means of time redundancy. This scheme requires low hardware cost and no performance penalty and achieves very high soft-error detection. However, this technique is not applicable to SRAM-based FPGA where soft errors could be latched into configuration memory and thus be transformed into permanent faults.

As a result of all these factors the development of suitable concurrent error detection scheme that meets overhead constraints and high error coverage is currently an important challenge. In this paper, we suggest a new solution that can be used in cost-sensitive mainstream applications to satisfy error-coverage requirements at minimum cost. Under this new paradigm, we present one particular non-intrusive, comparison-based, partially self-checking scheme. The proposed method achieves a very high error-coverage within the specified overhead constraints. This paper presents overhead estimations and results for MCNC 91 collection of benchmark combinational circuits, for the proposed algorithm.

The paper is organized as follows. Section 2 describes the adopted fault model and explains the basic idea of the proposed approach. Section 3 deals with some definitions and notation used in this paper. The proposed greedy single-pass algorithm used for under-approximation of the characteristic function is described in Section 4. The effectiveness of the proposed partial self-checking approach for some benchmarks is discussed in Section 5. Finally, we conclude in Section 6.

2. Partial Self-Checking Method

In this section we will first describe the adopted fault model and after that we will explain the basic idea of the proposed approach that deals with partial self-checking.

A. Error Model

We assume an unrestricted behavioral error model, i.e. an error model that is not defined through permanent or transient faults in the hardware, but rather in terms of erroneous behavior that such faults induce. Consider a combinational circuit with n inputs and m outputs that implements logic function f. For every input combination $X\in\{0,1\}^n$, we define an error-free response of the circuit as $Y=f(X)$, and a set of erroneous responses as $E=\{Y\in\{0,1\}^m \mid Y\neq f(X)\}$. According to our model, the CED circuit needs to distinguish between the error-free response, and the erroneous responses in the set E.

B. Function Checking with Duplication

Duplication-based CED techniques are of particular interest for our work. Figure 1 shows the *duplication with comparison* (DWC) scheme. The DWC (see Fig. 1(*a*)) consists of two modules, an *original function module* (OFM), and a *duplicated function module* (DFM), and an *m-bit comparator* (MBC). Both modules implement an identical n-input m-output logic function f. The MBC checks the existence of agreement between the outputs of both modules. If the outputs disagree, the MBC indicates an error. This method guarantees 100% error coverage in respect to any type of fault that affects the OFM. However, DWC's implementation results in hardware overhead greater than 100% [15].

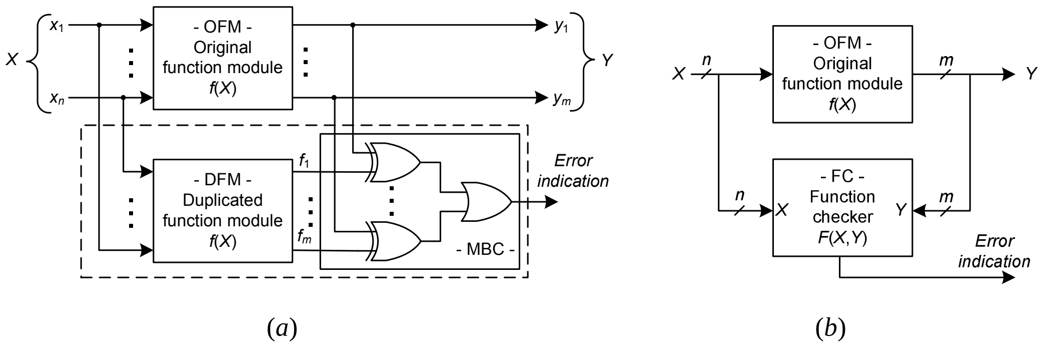


Fig. 1. Duplication with comparison: (*a*) standard representation; (*b*) alternative representation.

Figure 1(*b*) shows an alternative representation of the DWC concept that we have adopted in our work. In this proposal, two DWC's constituents, enclosed into dashed box of Fig.1(*a*), i.e. the DFM and the MBC, are implemented as a single module, referred to as a *function checker* (*FC*). The FC accepts both *n*-bit input, *X*, and *m*-bit output, *Y*, of the OFM, and generates a single output that indicates whether a given (*X*,*Y*) combination is valid or not. In other words, for each of 2^n possible input combinations, *X*, the FC is able to distinguish between error-free, $Y=f(\mathrm{X})$, and 2^m-1 erroneous responses, $Y \neq f(\mathrm{X})$, of the OFM.

We say that the FC implements *characteristic function*, *F*, of the original function *f* defined by $F(X,Y)=0$ if $Y=f(\mathrm{X})$, and $F(X,Y)=1$ when $Y \neq f(\mathrm{X})$. Note that in this case the meaning of the characteristic function is opposite to the natural one for which $F(X,Y)=1$ represents $Y=f(\mathrm{X})$ [16]. Having in mind that the FC in Fig. 1(*b*) detects any erroneous behavior of the OFM, we call this circuit a *total function checker* (TFC). It is evident that for 100% error coverage the area overhead induced by the TFC is identical to the area overhead (>100%) of the basic DWC scheme.

C. Partial Function Checking

In order to make a good compromise between the hardware overhead (<100%) and error-detecting potential (<100%), we introduce the concept of partial function checking. Given a logic function $Y=f(X)$ with characteristic function $F(X,Y)$, a *partial function checker* (PFC) in respect to f is a circuit that implements function $F^*(X,Y)$, where F^* represents an under-approximation of F. From one hand, the function F^* under-approximates F if the output of $F^*(X,Y)$ agrees with $F(X,Y)$ whenever $F(X,\mathrm{Y})=0$. From the other hand, the bit values for $F^*(\mathrm{X,Y})$ when $F(\mathrm{X,Y})=1$ can be arbitrary selected with a goal to reduce the complexity of F^*; i.e. how to obtain as much as greater (<100%) error coverage with minimal hardware overhead (<100%). We will denote this under-approximation as $F^* \subseteq F$.

The PFC recognizes all error-free (X,Y) combinations, but fails to detect a limited number of erroneous responses. Let $|g|$ denote the number of 1-minterms of function g, i.e. the number of inputs for which $g=1$. The *error coverage*, *cv*, of the PFC implementing F^* is defined as

$$cv = \frac{|F^*|}{|F|} \tag{1}$$

In other words, the error coverage corresponds to a ratio between the number of erroneous responses detected by PFC and the total number of erroneous responses of the OFM. The error coverage of a TFC ($F^*\equiv F$) is cv=1.0 (correspond to DWC), while the error coverage of an “empty” PFC ($F^*\equiv 0$) is cv=0 (CED is not implemented). Between these two end solutions there is a spectrum of PFC’s implementations of different costs and error coverage. The challenge, now, lies in devising algorithms that produce good under-approximations of the characteristic functions with minimal cost.

Example 1. In order to illustrate the basic idea of proposed method, let us analyze the example given in Fig. 2. The truth table for function f with two inputs, x_1 and x_2, and two outputs, y_1 and y_2, is presented in Fig. 2(a). Both the corresponding characteristic function, F, and its two under-approximations, F_1^* and F_2^*, ($F_2^* \subseteq F_1^* \subseteq F$), are defined by the truth table given in Fig. 2(b). Karnaugh maps of functions F, F_1^*, and F_2^*, as well as, the derived minimal sum-of-product expressions are given in Fig. 2(c), (d), and (e), respectively. Note that, for all four error-free conditions of the function f, the value in column F is 0 (see Fig. 2(b)). For example, when x_1=x_2=0, there must be y_1=1 and y_2=0 (Fig. 2(a)). Thus, F(0,0,1,0)=0. The other conditions that correspond to the same input combination, i.e. (0,0,0,0), (0,0,0,1), and (0,0,1,1), characterize erroneous behavior of the circuit implementing f. Therefore, their values in column F are equal to 1. The function F provides full error coverage with respect to function f, cv=1.0, with cost of 6 product terms with total of 16 literals (Fig. 2(c)). Under-approximations of F are obtained by replacing one or more of its 1-minterms with 0-minterms. If the elimination of 1-minterms is performed in a selective manner, so that the number of covering rectangles decreases, too, we will obtain F^* which is simpler than F. From one hand, as can be seen from Fig. 2(d), the function F_1^*, that includes two additional 0-minterms, has 4 product terms with 10 literals. From the other hand, the simplification of the characteristic function causes loss in error coverage, since erroneous responses represented by omitted 1-minterms become undetectable. Since F_1^* is able to detect 10 out of 12 possible erroneous conditions, its error coverage is reduced to cv=10/12=0.83. The function F_2^* is obtained by omitting additional two 1-minterms leading us to a solution with cv=0.75 and hardware cost of 3 product terms and 7 literals (Fig. 2(d)).

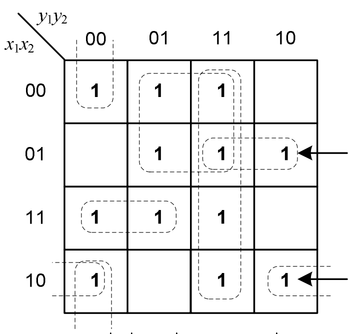
x_1	x_2	y_1	y_2
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	0

f: $y_1 = x_1'x_2' + x_1x_2$
$y_2 = x_1x_2'$

(*a*)

x_1	x_2	y_1	y_2	F	F_1*	F_2*
0	0	0	0	1	1	1
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
0	1	1	0	1	$\boxed{0}$	$\boxed{0}$
0	1	1	1	1	1	1
1	0	0	0	1	1	$\boxed{0}$
1	0	0	1	0	0	0
1	0	1	0	1	$\boxed{0}$	$\boxed{0}$
1	0	1	1	1	1	1
1	1	0	0	1	1	$\boxed{0}$
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	1	1

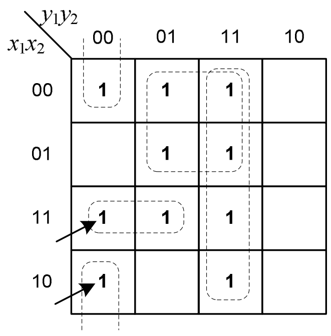
(*b*)



$$F = x_1x_2'y_2' + x_1'x_2 + x_1x_2y_1' + x_2'y_1'y_2' + y_1y_2 + x_2'y_1y_2$$

(*cv*=1.00, 6 prod. terms/16 literals)

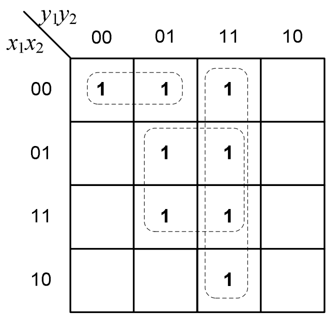
(*c*)



$$F_1^* = x_2'y_1'y_2' + x_1'y_2 + x_1x_2y_1' + y_1y_2$$

(*cv*=0.83, 4 prod. terms/10 literals)

(*d*)



$$F_2^* = x_1'x_2'y_1' + x_2y_2 + y_1y_2$$

(*cv*=0.75, 3 prod. terms/7 literals)

(*e*)

Fig. 2. Approximation of characteristic function example. (*a*) 2-input 2-output function; (*b*) truth table for characteristic function and two under-approximation functions; (*c*) Karnaugh map for *F*; (*d*) Karnaugh map for *F_1**; (*e*) Karnaugh map for *F_2**.

3. Definitions and notation

A cube-list is a data structure for representing and manipulating Boolean functions [16]. As an example, Fig. 3(*a*) illustrates a cube-list representation of the single-output function $f(x_1,x_2,x_3)$ defined by the sum-of-product (SOP) expression on the left. Each row of the cube-list is referred to as the *cube* and represents an algebraic product term. For example, the cube 1-0 in Fig. 3(*a*) represents the product term x_1x_3'. Note that, a cube may be interpreted as a subcube in Boolean *n*-cube, or a rectangle in the Karnaugh map, also.

f: $y=x_1'+x_2x_3+x_1x_3'$	0-- -11 1-0

(*a*)

f: $y_1=x_1x_2'+x_1'x_2$, $y_2=x_1+x_1'x_2$	10 10 01 11 1- 01

(*b*)

Fig. 3. Cube-list representation of: (a) single-output, (b) multiple-output function.

When a multiple-output function f: $(y_1, \ldots, y_m) = (f_1(X), \ldots, f_m(X))$ is represented in the cube-list notation, a *m*-bit tag field, called *output part*, is concatenated to the input partition of a cube. If the *i*th bit of the output part is a 1, the output y_i is occupied by the cube; otherwise this cube has no meaning for the value of this output. For example, Fig. 3(*b*) shows a two-output function represented in both SOP and cube-list notation. With this notation, the cover of output y_i is obtained by a union of cubes with 1 at *i*th position in output part.

A *minterm* is a cube in which every variable in the Boolean functions appears (e.g., 110, or 001). The minterm may be interpreted as a *vertex* in the Boolean *n*-cube, or as a *cell* in the Karnaugh map. A cube *q contains* (or *cover*) a minterm *r* if the literals of cube *q* are subset of the literals of minterm *r*. For example, the set of minterms covered by the cube 0-- is {000, 001, 010, 011}. Thus, a list of cubes represents the union of the minterms covered by each cube and is called a cubical cover of the minterms, or simply a *cover*. Given a cover *C*, the *essential minterms* of a cube $c \in C$ are those minterms that are in *C* and are not covered by any other cube in *C*. For example, essential minterms of cube --11, in the cover defined by the Karnaugh map in Fig. 1(*e*), are 0011 and 1011. The cube with all “-“, such as the cube ----, is referred to as the *universal cube* and denoted as Φ. The cube Φ covers all 2^n minterms of an n-dimensional Boolean space.

4. Approximation Algorithm

For a given arbitrary combinational circuit that implements a multi-output logic function f, let us derive the single-output logic function, $F*$. We then assume that F^* will be implemented by a partial function checker. During the process of deriving the logic function F^*, our goal is to achieve the specified error coverage (cv<1.0) while minimizing the cost of F^*. The cost metric is evaluated as the number of cubes (i.e. product terms) in a SOP expression of F^*.

In this section, we will describe a greedy, single-pass algorithm which we use for under-approximation of the characteristic function. The algorithm starts with deriving the cube-list of the characteristic function. It proceeds with successive cube elimination form the cube-list. The elimination process stops when the error coverage falls under a specified limit.

The algorithm is formally stated in Fig. 4. We assume that the function f of the OFM is known and is specified as SOP cube-list. From the Fig. 1(a) we conclude that the PFC is designed to detect operational errors of the already synthesized combinatorial circuit, i.e. OFM. Having this in mind, we assume that the function f is completely specified. In a case when the outputs of the circuit are not completely specified, preliminary two-level minimizations of f is performed in step 1 in order to eliminate don’t care conditions. Such minimization fixes all don’t cares at determined values.

In step 2, using the cube-list based procedure outlined in Subsection 4.*A*, we derive the characteristic function, *F*. A complement of function *f* is determined in this step. For this task, we use `espresso` with option `–epos` switched on [17]. In order to, initially, reduce the number of cubes, the characteristic function is minimized by using `espresso` (step 3). The number of minterms covered by *F* is determined in step 4. For more details concerning this step, see Subsection 4.*B*. In step 5, F^* is initially set to *F*, and its error coverage, $cv(F^*)$, is set to 1.0. Prior to the start with approximation (step 6) the number of essential minterms for each cube in F^* is calculated.

Steps 7 to 12 represent the crucial steps of the algorithm. Until the error coverage drops below a specified limit, cv_{goal}, in one-by-one manner, cubes of F^* are removed. In step 7, we decide which of the cubes will be removed. Among the cubes in the current cover F^*, a cube with minimal number of essential minterms will always be eliminated. If there is a tie between the cubes, we choose one at random. Note that by removing a cube from a cover, the essential minterms of that cube will actually be eliminated from the cover, only. Thus, in each iteration, the algorithm lowers the cost of F^* for one cube, but at some time maintains losses in error coverage as minimal as possible. The selected cube is removed from F^* in step 8, and error coverage of F^* is re-calculated in step 9. If error coverage is still above the target, the algorithm updates the number of essential minterms of the remaining cubes and repeats the cube elimination procedure. Note that during updating a number of essential minterms in step 11, we use the fact that this number could be changed for those cubes that intersect with the cube removed in the present iteration, only.

Finally, when the error coverage becomes lower then cv_{goal}, the algorithm exits the loop. In step 13, it applies two-level logic minimizer on F^* (step 13) in order to reduce the number of covering cubes in its last attempt.

In the sequel, we give a more detailed explanations of that how in step 2 the characteristic function is derived, and how in steps 4, 6 and 11 the minterms are counted.

```
Input:    n-input, m-output Boolean function f given in SOP cube-list form
          target error coverage, $cv_{goal}$
Output: Boolean function $F^*$ given in SOP cube-list form
/* $|F|$ = size (number of minterms) of $F$
   $em(c)$ = number of essential minterms of cube $c$
   $cv(F^*)$ = error coverage of $F^*$
   $\ominus$ - disjoint sharp operator
   $\setminus$ - set difference operator */
1.  Minimize $f$.
2.  Compute the characteristic function $F$ of $f$.
3.  Minimize $F$.
4.  Set $|F| = 2^{m+n} - |\Phi \ominus F|$ .
5.  Set $F^* = F$. Set $cv(F^*) = 1.0$.
6.  For each cube $c \in F^*$ set $em(c) = |c \ominus (F^* \setminus c)|$.
7.  Find $q \in F^*$ with smallest $em(q)$.
8.  Set $F^* = F^* \setminus q$.
9.  Set $|F^*| = |F^*| - em(q)$. Set $cv(F^*) = \frac{|F^*|}{|F|}$ .
10. If $cv(F^*) < cv_{goal}$ goto step 13.
11. For each cube $r \in F^*$ and $q \cap r \neq \varnothing$ set $em(r) = |r \ominus (F^* \setminus r)|$.
12. Goto step 7.
13. Minimize $F^*$.
14. Return $F^*$.
```

Fig. 4. Algorithm for under-approximating characteristic function.

A. Deriving characteristic function

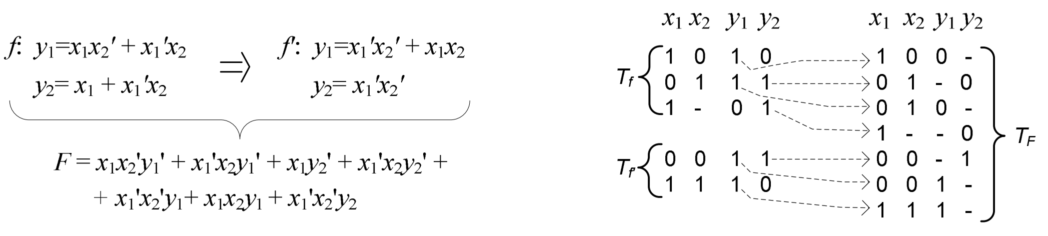
For a given Boolean function $f(X)$: $y_i=f_i(X)$, $i=1, \ldots, m$, the characteristic function, $F(X,Y)$ of f can be defined in terms of Boolean operations as follows (see Fig 1(*a*)):

$$F(X,Y) = \sum_{1 \le i \le m} (f_i(X) \oplus y_i), \tag{2}$$

or

$$F(X,Y) = \sum_{1 \le i \le m} (f_i(X) y_i^{'} + f_i^{'}(X) y_i) \tag{3}$$

According to Eq. (3) the SOP expression of the characteristic function F is obtained as follows. First, for each i, $i=1,\ldots,m$, we extend every product term of f_i with literal $y_i^{'}$, and every product term of $f_i^{'}$ with y_i. Then, we OR together all extended product terms in a single SOP expression F, as illustrated by an example in Fig. 5(a).



(a) (b)

Fig. 5. Deriving characteristic function, an example: (a) SOP-based method; (b) cube-list method.

The Eq. (3) suggests to the manner how, starting from a cube-list, we can derive the characteristic function. Figure 5(*b*) illustrates this cube-list based procedure for the same function used in Fig. 5(*a*). For a given cube-list T_f of the function f, we need to generate first the cube-list $T_{f'}$ of f'. After that we process both cube-lists, T_f and $T_{f'}$, cube-by-cube, and create one cube in list T_F of the characteristic function F, for each 1 encountered in the output part of cubes that belongs to $T_f \cup T_{f'}$. In the cube-list T_f from Fig. 5(*b*) consider now the cube 10 10. Since this cube has a single 1 in its output part,

y_1=1, it induces one cube in *F*, 100-. In this cube, input values x_1 and x_2 are copied, y_1 is lowered to 0, y_2 is raised to -. The cube 01 11 in *f*, produces two cubes in *F*, one for y_1=1, 010-, and the other for y_2=1, 01-0. Similarly, each literal equal to 1 in the output part of *f'*, produces one cube in *F*. But, now, the value of this literal in the new cube is equal to 1, while all other output literals are raised to -.

B. Counting number of minterms

The *size* of the cube *c*, denoted as |*c*|, corresponds to the number of minterms that the cube contains. The size of *n*-variable cube *c* with *k* literals is $|c|=2^{n-k}$. Let $F=\{c_i \mid i=1,\ldots,k\}$ be a cover. When cubes in *F* are disjoint (i.e. when no two cube cover a common minterm, $c_i \cap c_j = \varnothing$ for $i \neq j$), the number of minterms in *F* can be simply calculated by the formula:

$$|F| = \sum_{1 \le i \le k} |c_i| \tag{4}$$

In other words, if cubes identify non-overlapping pieces of the *n*-dimensional Boolean space, then the size of the cover can be found by simply summing up the sizes of different pieces. However, in general, this will not be the case. An approach for computing the number of minterms in a general cover is based on usage of the so called *disjoint sharp* operator [16]. This operator, denoted as $\ominus$, is used to compute the difference between two covers. For two covers *C* and *D*, difference $C \ominus D$ defines the set of pair-wise disjoint cubes that cover the minterms covered by *C* but not by *D*. Using this operation, we can count the number of minterms in a cover *F*, as follows. First, we compute the complement cover of *F*, as $F' = \Phi \ominus F$, and then we apply Eq. (4) in order to obtain the size of *F'*. Since *F* and *F'* divide the Boolean space Φ into two disjoint partitions, the size of the *F* is $|F|=|\Phi| - |F'|$.

Similarly, when counting the number of essential minterms in a cube $c \in F$, we first employ a disjoint sharp operation in order to cover these minterms by a set of disjoint cubes, $c \ominus (F / c)$, and after that we sum up their sizes.

5. Experimental Results

To demonstrate the effectiveness of the proposed partial self-checking approach, we carried out several experiments using subsets of the MCNC 91 collection of benchmark combinational circuits. Table 1 presents the results. Under the first major heading, details about each chosen circuit related to: number of inputs (*In*), number of outputs (*Out*), and number of product terms (*Pt*) after two-level logic minimization are given. Under the second and third major headings for each circuit we report the results concerning characteristic function and its five approximations obtained by the method proposed in this paper. The heading *F* relates to the DWC, while headings F^* deals with five approximations of relatively high error coverage ($cv \geq 85\%$). These results include: number of product terms in the minimized SOP expression (*Pt*) and the percentage of product terms overhead in respect to the original circuit (*Ov*).

Presented results indicate that for some benchmarks the proposed approach is very effective when high error coverage and low hardware overhead are required. So, for example, the original “alu4” benchmark circuit requires 545 product terms for implementation, while only 89 product terms (i.e. 16% percent of the original circuit cost) cover 99% of all erroneous responses. Let us note that there are other benchmark circuits, referred as “5xp1”, “b12”, “misex3c”, and “table3” that can be easily approximated, also. However, the experiments made on some other benchmark circuits, reveal circuits that are hard to approximate. For example, in order to cover 85% of erroneous responses of the benchmark circuit “rd73”, we need to invest in partial function checker almost 71% of the original circuit cost. Roughly speaking we can say that, by using the proposed method, on average, one half of the original circuit cost is required to achieve 90% error coverage. It is evident from the results that the proposed scheme reduces the area overhead while providing high error coverage in all the cases.

Table 1. Results for MCNC 91 benchmark circuits.

Circuit (*f*)				Characteristic function - *F*		Approximated characteristic function - F^*									
				cv=1.0		*cv*=0.99		*cv*=0.98		*cv*=0.95		*cv*=0.90		*cv*=0.85	
Name	In	Out	Pt	Pt	Ov	Pt	Ov	Pt	Ov	Pt	Ov	Pt	Ov	Pt	Ov
5xp1	7	10	65	82	126	32	49	23	35.3	15	23.1	10	15.4	8	12.3
9sym	9	1	86	160	186	151	175	146	169	133	155	105	122	89	103
alu4	14	8	545	284	52	89	16.3	61	11.2	34	6.2	19	3.5	12	2.2
b12	15	9	43	87	202	25	58.1	19	44.2	13	30.2	9	20.9	7	16.2
clip	9	5	120	123	102	94	78.3	81	67.5	58	48.3	39	32.5	29	24.2
con1	7	2	9	17	188	16	177	15	166	14	155	11	122	9	100
ex1010	10	10	284	1052	370	577	203	402	142	334	117	246	86.6	194	78.8
inc	7	9	30	64	213	29	96.7	23	76.6	15	50	10	33.3	7	23.3
misex1	8	7	12	36	300	20	166	17	142	13	108	9	75	8	66.7
misex3c	14	14	197	387	196	40	20.3	31	15.7	20	10.1	13	6.6	10	5
rd53	5	3	31	45	145	42	135	40	129	33	106	26	83.9	21	67.7
rd73	7	3	127	176	138	164	129	157	123	130	102	106	83.5	90	70.8
rd84	8	4	255	346	136	307	120	269	105	200	78.4	120	47	83	32.5
sao2	10	4	58	80	138	36	62	24	41.4	15	25.8	10	17.2	7	12.1
squar5	5	8	25	47	188	27	108	21	84	15	60	10	40	7	28
t481	16	1	481	841	175	566	118	475	98.7	329	68.4	219	45.5	164	34.7
table3	14	14	157	430	274	55	35	39	24.8	24	15.3	14	8.9	10	6.4
Avr.(%)					**184**		**102**		**86.8**		**68.1**		**49.6**		**40.2**

The run time of the algorithm is largely dependent on the number of the cubes in the final solution. This dependence results because the number of disjoint sharp operations (step 11 in Fig. 4) is proportional to the square of the number of cubes in the list. In addition, the complexity of disjoint sharp operation grows exponentialy with the number of variables. For benchmarks with the number of inputs and outputs less the 16, such as “5xp1”, “9sym”, “clip”, and “inc”, the execution time of the algorithm is less the 1s on PC Pentium IV 2GHz machine. For medium size benchmark circuits, with the number of inputs and outputs up to 32, such as “b12”, “ex1010”, “misex3c”, and “table3”, the execution time is within the range from 30s to 30min, depending on the number of cubes.

6. Conclusion

In this paper, we have described a new approach for combinational circuits with concurrent error detection design that exploits the trade-off between the conflicting objectives of low hardware overhead and high error coverage. The proposed, partially self-checking scheme, employs comparison-based approach wherein functionality of standard composition of duplicated logic and equality comparator is first expressed as a single-output, characteristic function, which is after that simplified by utilizing procedure for Boolean function approximation. Cube-list based heuristic approximation procedure has been proposed. It extracts another function from a characteristic function, such that it preserves most minterms with substantial reduction in the number of cubes.

A set of benchmark circuits from MCNC 91 were used to demonstrate the efficiency of the proposed scheme. Results which show that this method can significantly reduce the area overhead required for concurrent error detection in two-level circuits while still detecting majority of circuit’s erroneous responses were presented here. In addition, we also show some difficult cases form the MCNC 91 benchmark circuits where hardware efficiency can be very hard to obtain. Nevertheless, we show that the proposed partially self-checking approach can be effectively applied to most random logic.

References

1. R. Ronen, A. Mendelson, K. Lai, S.-L. Lu, F. Pollack, and J.P. Shen, Coming Challenges in Microarchitecture and Architecture, IEEE Proceedings 89 (3), 2001, 325-340.
2. K. Mohanram, and N.A. Touba, Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits, in Proc. of IEEE International Test Conference, 2003, 893-901.
3. P.K. Lala, Self-Checking and Fault-Tolerant Digital Design, Morgan Kaufmann Publishers, San Francisco, 2001.
4. D.K. Pradhan, Fault-Tolerant Computing: Theory and Techniques, Vol. I, Prentice Hall, New Jersey, Englewood Cliffs, 1986.
5. K. Mohanram and N. A. Touba, Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits, in Proc. International Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS), 2003, 433-440.
6. K. De, C. Natarajan, D. Nair, and P. Banerjee, RSYN: A system for automated synthesis of reliable multiplevel circuits, IEEE Trans. on VLSI Systems, 2 (2), 1994, 184-195.
7. N.K. Jha, and S.J. Wang, Design and Synthesis of Self-Checking VLSI Circuits, *IEEE Trans. on CAD*, 12(6), 1993, 878-887.
8. S. Mitra, and E.J. McCluskey, Which concurrent error detection scheme to choose?, in *Proc. IEEE International Test Conference*, 2000, 985-994.
9. C.M. Jones, S.S. Dlay and R.N.G. Naguib, Berger Check Prediction for Concurrent Error Detection in the Braun Array Multiplier, *Microelectronics Journal*, 27(8), 1996, 745-75.
10. D. Das, and N.A. Touba, Synthesis of Circuits with Low-Cost Concurrent Error Detection Based on Bose-Lin Codes, *Journal on Electronic Testing: Theory and Applications (JETTA)*, 15(1/2), 1999, 145-155.
11. M.A. Narouf, and A.D. Friedman, Efficient Design of Self-Checking Checker for any m-Out-of-n Code, *IEEE Trans. on Computers*, C-27(6), 1978, 482-49.
12. D. Das, and N.A. Touba, "Weight-Based and Their Application to Concurrent Error Detection of Multilevel Circuits", In *Proc. 17th IEEE VLSI Test Symposium*, Dana Point, CA, 1999, 370-376.
13. Touba N.A., McCluskey E.J., Logic Synthesis of Multilevel Circuits with Concurrent Error Detection, *IEEE Transaction on CAD*, 16(7), 1997, 783-789.
14. M.K. Stojcev, G. Lj. Djordjevic, and T.R Stankovic, Implementation of self-checking two-level combinational logic on FPGA and CPLD circuits, *Microelectronics Reliability*, Elsevier, 44(1), 2004, 173-178.
15. S. Mitra, and E.J. McCluskey, Design of Redundant Systems Protected Against Common-Mode Failure", in Proc. of 19th IEEE VLSI Test Symposium, 2001, 190-197.
16. S. Devadas, A. Ghosh, and K. Keutzer, Logic Synthesis, McGraw-Hill, New York, 1994.

17. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, MA, Boston, 1984.

18. L. Anghel, M. Nicolaidis, Cost Reduction and Evaluation of a Temporary Faults Detecting Technique, in Proc. of Design, Automation and Test in Europe (DATE '00), 2000, 592-598.

<u>Figure Captions:</u>

Fig. 1. Duplication with comparison: (*a*) standard representation; (*b*) alternative representation.

Fig. 2. Approximation of characteristic function example. (*a*) 2-input 2-output function; (*b*) truth table for characteristic function and two under-approximation functions; (*c*) Karnaugh map for *F*; (*d*) Karnaugh map for F_1*; (*e*) Karnaugh map for F_2*.

Fig. 3. Cube-list representation of: (*a*) single-output, (*b*) multiple-output function.

Fig. 4. Algorithm for under-approximating characteristic function.

Fig. 5. Deriving characteristic function, an example: (a) SOP-based method; (b) cube-list method.

<u>Table Captions:</u>

Table 1. Results for MCNC 91 benchmark circuits.