Non-redundant Scheme for Arbitrary Error Detection in Combinational Circuits

Osnat Keren, Ilya Levin and Mark Karpovsky

***Abstract*— The paper deals with synthesis technique for designing circuits with on-line errors detection. We present a new technique of designing a concurrently checking functional circuit by partitioning the circuit into two independent sub-circuits. The technique does not require any redundant coding variables. Instead, we propose to utilize some input variables. These variables are transferred directly into the checker providing the self-checking property for arbitrary errors (not necessarily unidirectional). A method for constructing the overall system is presented. Benchmark results are presented and show the efficiency of the proposed approach.**

***Index Terms*— Concurrent error detection, on-line checking, combinational circuits, partitioning**

I. Introduction

Progress in the microelectronics industry leads to increase of complexity of VLSI schemes and components. The number of transistors in the VLSI schemes has already reached millions, and in some cases has risen even higher. Shrinkage of a device size and reduction of power supply levels, as well as the increase in operating speed has resulted in reduced noise margins [17]. The failures phenomena, together with the need for higher reliability of complex digital systems, are of special interest. As the microelectronics industry moves toward deep sub-micron technologies, systems designers have become increasingly concerned about the reliability of future devices that will have propagation delays shorter than the duration of transient pulses induced by radiation attacks. They are also concerned about smaller transistors, which will be more sensitive to the effects of electromagnetic noise, neutron and alpha particles, which may cause transient faults, even in fully tested and approved circuits [1]. Most of faults that occur in VLSI circuits and systems are transient/intermit in nature. The self-checking property allows both the transient and permanent faults to be detected, thus preventing data contamination.

Systematic error-detecting coding is one of the most effective instruments for concurrent error detection. This type of coding often utilize separate codes, since such codes allow preserving informational bits of the binary words to be coded, while complementing the binary words by check bits. The coded binary words form a set A of codeword. The set A can be defined either by a list of the codewords, or by a specific property distinguishing the codewords from non-codewords. In order to ensure that the codewords differ from non-codewords, each non-codeword, formed at an output of a scheme instead of a codeword a due to a specific type of fault, should either not to belong to the set A, or to be equal to the correct word in A. Models of distortion of codewords are usually built taking into account characterizing features of the stream of faults and of the scheme under checking. As it is accepted in relevant papers, we will consider that faults are manifested by pins signals of ”0” or ”1” on input or output contacts of logical elements forming the scheme to be checked. The faults can be temporary or permanent. It is traditionally accepted that a time interval between occurrences of two adjacent faults is sufficient for coping with the earliest fault. Therefore, only a single fault can present simultaneously in a scheme under checking. This fact is usually considered when building models of acceptable distortions of output codewords. Most of relevant publications use the following two models of distortions.

The first model is based on an assumption that a system of functions, which reflects conversion of information in a scheme under check, is monotonous. For example, schemes that do not comprise inverters satisfy the mentioned assumption. In such schemes, any single fault may only result in so-called unidirectional faults of output code words [6]. The Berger code [2], and sometimes the Smith code [15] are used for detecting unidirectional errors. A number of algorithms are known [4], [5], [14], [9], which allow converting an arbitrary scheme in such a manner that the Berger code could be used for its checking. To this end, the scheme can be modified in such a way, that only its input variables become negated [4]. The works [5], [14] propose algorithms of converting the scheme under check by duplication of some of its elements. The paper [9] describes a combination of several approaches.

The second model is based on an assumption that a number of distorted bits in a codeword is not greater than a predetermined threshold t. The paper [12], based on a study of a large number of benchmarks, shows that in most of the cases a single fault results in errors in two or less bits of a codeword.

The majority of the known concurrent checking schemes assumes that a set of output codewords of the functional circuit to be checked is complete, i.e., any binary vector is a codeword. However, it is often reasonable to construct a so-called context-oriented concurrent checking scheme, where: a) the number of possible codewords, is much smaller than 2^k , where k is the width of the output vector, and b) the set of possible codewords is known in advance. The context-orientation has some advantages in comparison with the universal orientation. Namely, it allows utilizing the redundancy of the circuit’s output codewords, which is an intrinsic feature of such circuits. One of the ways of utilizing the redundancy is by dividing the functional circuit into a number of separate independent sub-circuits. Each of these sub-circuits implements its own subset of output signals. Since the sub-circuits have no common elements, any single fault may result

in errors only in a subset of the output signals.

The context-orientation was studied in [7]. A so-called Sum-of-minterms (SOM) checker was proposed. The SOM checker tests whether an output word belongs to the set of possible code-words of the circuit to be checked. In [8], the authors developed a specific architecture for checking occurrence of unidirectional errors in sequential circuits without introducing any redundant coding variables. Instead of the encoding they proposed a match-detector based self-checking architecture. This architecture uses signals of logic products of the functional unit for providing the self-checking property. Signals of these products form additional inputs of the checker. Partitioning a functional circuit for mutually checking components was proposed in [10] as another way for exploring the context-orientation. The authors examined a two-block partition, minimizing the number of encoded variables in a concurrent checking scheme that detects any arbitrary errors.

In the present paper, the idea of a circuit's partitioning for the context-oriented concurrent checking is studied from a different point of view. As in [8], we don't use any redundant coding for the output vectors. Instead, we propose to transfer the input variables (and not the products) into the checker. Actually, these input variables are used instead of the conditional coding variables. A method for constructing a system that provides the self-checking property for arbitrary errors is the main contribution of the paper. We show that the partitioning of the initial circuit into independent sub-circuits followed by choosing the optimized set of input variables is efficient for detecting both unidirectional and arbitrary errors.

The paper is organized as follows: Section II includes basic definitions, and recalls related work on on-line testing for arbitrary errors. Section III presents the suggested structure, and Section IV contains experimental results. Section V concludes the paper.

II. Preliminaries

A. Context-oriented coding

Consider a functional unit that has m inputs and k outputs. The logic unit can be represented as a multi-output function $Y = f(X)$ where $X = (x_{m-1}, \ldots, x_0)$ and $Y = (y_{k-1}, \ldots, y_0)$. In this paper, the binary output vectors are referred to as *information words*. Assume that the logic unit can produce only M distinct information words out of the 2^k possible combinations, that is, $M < 2^k$.

In order to detect a single fault in the system, conventional methods encode the information words by adding redundancy bits. Namely, each information word $Y_i = (y^{(i)}_{k-1}, \ldots, y^{(i)}_0)$ is encoded to a codeword $Z_i = (z^{(i)}_{n-1}, \ldots, z^{(i)}_0)$ of length $n \geq k$. The set of codewords $\{Z_i\}_{i=1}^{M}$ forms a code.

Definition 1: A code is called *systematic* , if there are k fixed positions $\{j_s\}_{s=0}^{k-1}$ such that $z^{(i)}_{j_s} = y^{(i)}_s$ for all $1 \leq i \leq M$ and $0 \leq s < k$. The remaining $r = n - k$ position carry the redundancy.

Clearly, systematic codes are preferable since they allow extracting an information word Y_i out of a codeword Z_i without additional processing. For faults that cause *unidirectional* errors, the commonly used codes are the systematic Berger code, [2], which adds $r = \lceil log_2(k+1) \rceil$ redundancy bits, and the non-systematic $w - out - of - n$ code, [3], for which the M information words are mapped to binary vectors of length n and of Hamming weight w, where, $M \leq \binom{n}{w}$. The context-oriented systematic Smith code [15] allows, in some cases, to reduce the number of redundancy bits.

The assumption that a fault causes unidirectional errors allows to implement the functional unit as a single circuit. However, in cases where a fault may cause any type error (not necessarily unidirectional), the fault may not be detected in functional units that are implemented as a single circuit. In order to detect an arbitrary fault the functional unit should be implemented by at least two independent circuits [11].

Coding schemes for such a case, were discussed in [5], [11], [13], [16]. In this paper we assume that the functional unit is implemented as two independent circuits. Without loss of generality, we assume that the first circuit realizes the first n_1 bits of Z, that is, $c_1 = (z_{n_1-1}, \ldots, z_0)$, and the second circuit realizes the remaining $n_2 = n - n_1$ bits, $c_2 = (z_{n-1}, \ldots, z_{n_1})$. The overall system is *fault-secure* in respect to a single fault in one of the circuits, if either, a fault maps a codeword on itself, or it maps a codeword to a non codeword [6]. In [11] it was shown, that iff

$$c_2 = f_1(c_1), \text{ and, } c_1 = f_2(c_2), \tag{1}$$

then the functional unit is fault secure in respect to arbitrary errors in one of the two circuits.

B. SOM checker for context-oriented codes

In [7] the authors introduced a Sum Of Minterms (SOM) checker for a context-oriented code in respect to unidirectional errors. The SOM checker is based on dividing the set of M codewords into two disjoint sets, Π_0 and Π_1. The checker consists of two independent circuits, each implements a function

$$R_i = \vee_{Z_j \in \Pi_i} m(Z_j) \quad , \quad i = 0, 1,$$

where $m(Z_j)$ is a minterm of the variables $z_0, \ldots z_{n-1}$ that corresponds the codeword Z_j. In normal operation (i.e when there is no fault in the system), the output of the checker is $(R_0 R_1) \in \{(01), (10)\}$; Whereas, at the presence of a single fault in one of the circuits, the output of the checker is $(R_0 R_1) \in \{(00), (11)\}$. Inherently, the SOM checker is fault secure but it is not self testing [7]. The structure of the overall system is shown in Figure 1. Experimental results show that for $M < 2^k/3$, the system in [7] has smaller implementation cost than a solution based on duplication of the functional unit combined with a dual-rail checker.

Figure 2, shows a structure of a system that is based on [7], and that can detect any type of single fault in one of its circuits. The system combines a functional unit that is implemented as two independent circuits with a SOM checker. The functional unit produces context-oriented systematic codewords having $n_1 = k_1$ and $n_2 = k_2 + r$, where $k_1 + k_2 = k$.

III. Suggested scheme - redundancy free solution

A. Fault detection by routing the inputs to a SOP checker

In this paper we suggest a new approach for detecting an arbitrary single fault. Instead of encoding an information

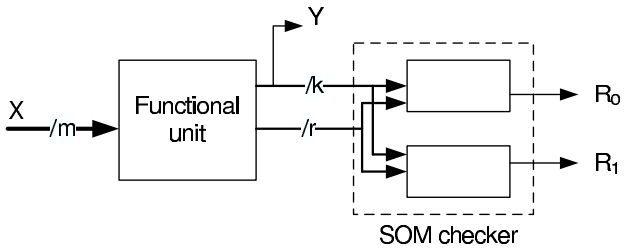

Fig. 1. On-line checking scheme for unidirectional errors

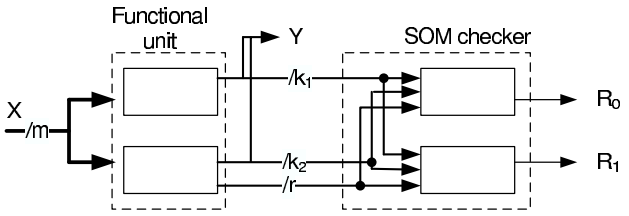

Fig. 2. On-line checking scheme for arbitrary errors

word Y_i into a codeword Z_i by adding r redundancy bits, we suggest to use the information bits as they are (uncoded), and rout the x's as inputs to the checker. We will show that this solution is simpler and has a lower implementation cost than the duplication based solutions.

The suggested structure is shown in Figure 3. The functional unit is implemented as two independent circuits. The first circuit realizes k_1 bits of Y, that is, $c_1 = (y_{j_{k_1-1}}, \ldots, y_{j_0})$; The second circuit realizes the remaining $k_2 = k - k_1$ bits, that is, $c_2 = (y_{j_{k-1}}, \ldots, y_{j_{k_1}})$. The output of the functional unit is denoted by $\hat{Y} = (c_2, c_1)$. Obviously, there is a one-to-one mapping between Y_i and $\hat{Y}_i$. The input variables X together with the $\hat{Y}$ enter a (Sum Of Products) SOP checker.

As in [11], a partition of the information bits $\{y_j\}_{j=0}^{k-1}$ into two sets is done with the aim to minimize the number of y's in c_1 that completely specify the information words; Namely,

$$Y = \hat{f}(y_{j_{k_1-1}}, \ldots, y_{j_0}) = \hat{f}(c_1). \tag{2}$$

The remaining information bits (if any) form c_2. Consequently, the circuit that realizes c_2 is fault secure. Note that this partition may be not the optimal in terms of the implementation cost, but it is simple to construct and implement.

Definition 2 (Distance): The distance between two words $\hat{Y}_i = (c_2^{(i)}, c_1^{(i)})$ and $\hat{Y}_j = (c_2^{(j)}, c_1^{(j)})$ is

$$d(\hat{Y}_i, \hat{Y}_j) = |\{k | c_k^{(i)} \neq c_k^{(j)}),\ k = 1, 2\}|.$$

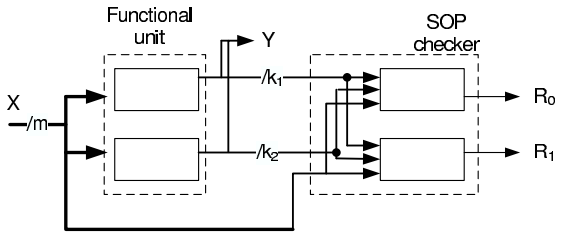

Fig. 3. Proposed structure: non redundant functional unit with SOP checker

	000	001	011	010	110	111	101	100
00	Y4	Y1	Y5	Y6	Y3	Y3	Y5	Y2
01	Y5	Y6	Y6	Y5	Y1	Y5	Y5	Y1
11	Y2	Y2	Y5	Y5	Y5	Y6	Y5	Y5
10	Y5	Y1	Y5	Y5	Y5	Y6	Y2	Y5

Fig. 4. K-map for the functional unit in Example 1

Definition 3 (Minimum distance): The minimum distance between the words in the set $\mathcal{Y} = \{\hat{Y}_i\}_{i=1}^{M}$ is

$$d_{min} = min_{\hat{Y}_i, \hat{Y}_j \in \mathcal{Y},\ i \neq j} d(\hat{Y}_i, \hat{Y}_j).$$

Clearly, any fault in the circuit that realizes c_1 is detectable if and only if all the codewords are of distance two from each other. Nevertheless, in some cases, it is not possible to find a partition that leads to a d_{min} that equals two. The following example illustrates such a case.

Example 1: Consider a functional unit that has five inputs $X = (x_4, \ldots x_0)$ and five outputs $Y = (y_4, \ldots y_0)$. Assume that the system produces only $M = 6$ information words $\{Y_i\}_{i=1}^{6}$ out of the possible 2^5 combinations, and,

$$\begin{array}{lll} Y_1 = (01011) & , & Y_2 = (00001) \\ Y_3 = (00101) & , & Y_4 = (10111) \\ Y_5 = (11010) & , & Y_6 = (11111). \end{array}$$

The functionality of the system is given in Figure 4 in a Karnaugh-like map; the labels on the columns correspond to the input variables (x_4, x_3, x_2), and the rows to (x_1, x_0).

The partition $c_1 = (y_4, y_3, y_2)$ and $c_2 = (y_1, y_0)$ fulfills Eq. 2. The distance between $\hat{Y}_1$ and $\hat{Y}_2$ is

$$d((010, 11), (000, 01)) = 2.$$

The distance between $\hat{Y}_1$ and $\hat{Y}_4$ is

$$d((010, 11), (101, 11)) = 1.$$

Thus, the minimum distance between the set of words is one. Consequently, there may be faults in the circuit that realizes c_1 that may no be detected. Note that since $k = 5$ it is not possible to find a partition that leads to a d_{min} that equals two. Hence, it is not possible to detect a fault in the circuit that realizes c_1 unless *'side-information'* is given to the checker.

B. SOP checker construction

In this subsection we present a checker that is based on products rather on minterms. We assume that the functional unit is implemented as two independent circuits, and that the partition to c_1 and c_2 fulfills Eq. 2. The inputs to the checker are X and $\hat{Y}$.

Definition 4 (Characteristic function): Let $\mathcal{Y} = \{\hat{Y}_i\}_{i=1}^{M}$ be the set of M words produced by the functional unit. The characteristic function $g_i(X)$ of Y_i, in respect to $\mathcal{Y}$, is

$$g_i(X) = \begin{cases} 1 & f(X) = Y_i \\ 0 & f(X) = Y_j \ and \ d(\hat{Y}_i, \hat{Y}_j) = 1 \\ \phi & otherwise \end{cases} \tag{3}$$

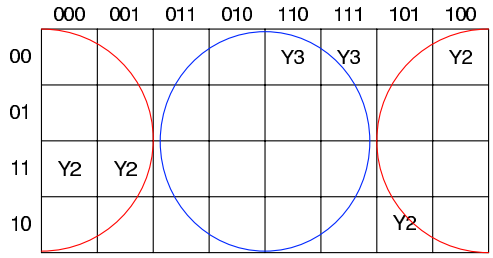

Fig. 5. K-maps for Example 2: characteristic functions of Y_2 and Y_3.

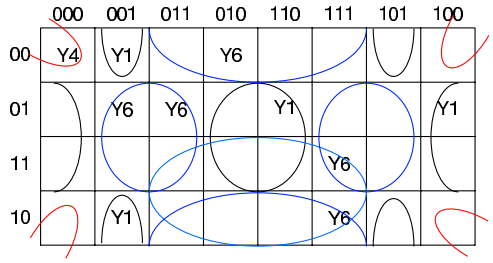

Fig. 6. K-maps for Example 2: characteristic functions of Y_1, Y_4 and Y_6.

where ϕ stands for don't care.

Example 2: The codeword $\hat{Y}_2$ in Ex. 1, is of distance one from the word $\hat{Y}_3$, and it is of distance two from all the remaining words. The Karnaugh-like map given in Figure 5 shows the combination of X's that produce Y_2 and Y_3. The empty bins in the map correspond to the input combinations for which the characteristic functions $g_2(X)$ and $g_3(X)$ are not specified. Clearly, the simplest expressions for the characteristic functions are $g_2(X) = x_3'$ and $g_3(X) = x_3$.

Note that the characteristic functions are not necessarily orthogonal (disjoint). That is, $\sum_X g_i(X)g_j(X) \geq 0$. See for example the characteristic functions $g_1(X)$ and $g_6(X)$ that are shown in Figure 6. Although the characteristic functions are not orthogonal, they can be used to detect a single fault by a SOP checker as states in Theorem 1 below.

A SOP checker is based on dividing the set $\mathcal{Y}$ of into two non-empty and disjoint sets, Π_0 and Π_1. The SOP checker consists of two independent circuits. Each circuit implements the function

$$R_i = \vee_{\hat{Y}_j \in \Pi_i} m_j(Y)g_j(X) \quad , \quad i = 0, 1,$$

where m_j is the minterm in the variables $y_0, \ldots y_{k-1}$ that represents the word $\hat{Y}_j$. Indeed m_j can be written as a product of two minterms $m_{j,1}$ and $m_{j,2}$ that represent the sub-words $c_1^{(j)}$ and $c_2^{(j)}$ which compose the word $\hat{Y}_j$,

$$m_j(Y) = m_{j,1}(c_1)m_{j,2}(c_2).$$

We now show that the overall system is fault secure in respect to any single fault that may occur in one of the four independent circuits: the two circuits of the functional unit and the two circuits that realize the checker.

Theorem 1: In a fault free circuit, the output of the checker is $(R_0R_1) \in \{(01), (10)\}$. In the presence of a single fault in one of the two circuits of the functional unit, the output is $(R_0R_1) \in \{(00), (11)\}$.

Proof: Without loss of generality, assume that the input to the system is $X = a$, $f(a) = Y_a$, and $Y_a \in \Pi_1$. Consider three cases:

- There is no fault in the system. Then,
$$m_j(Y_a)g_j(a) = \begin{cases} 1 & j = a \\ 0 & otherwise \end{cases} .$$
Thus, $R_0 = 0$ and $R_1 = 1$, and therefore, $R_0 = R_1'$.
- There is a single fault in the circuit that realizes c_1. Then, instead of the word $\hat{Y}_a = (c_2^{(a)}, c_1^{(a)}) = (\alpha, \beta)$, the functional unit may produce the word (α, γ) where $\beta \neq \gamma$.
If there exists a word $\hat{Y}_b \in \mathcal{Y}$ such that $c_2^{(b)} = \alpha$ and $c_1^{(b)} = \gamma$, then the characteristic functions of $\hat{Y}_a$ and $\hat{Y}_b$ satisfy $g_a(a) = 1$ and $g_b(a) = 0$. Therefore,
$$m_{j,2}(\alpha)m_{j,1}(\gamma)g_j(a) = \begin{cases} 1 \cdot 0 \cdot 1 = 0 & j = a \\ 1 \cdot 1 \cdot 0 = 0 & j = b \\ 0 \cdot 0 \cdot \phi = 0 & otherwise \end{cases} .$$
Thus, the output of the checker will be $R_0 = R_1 = 0$.
- There is a single fault in the circuit that realizes c_2. Then, instead of the word $\hat{Y}_a = (c_2^{(a)}, c_1^{(a)}) = (\alpha, \beta)$, the functional unit may produce the word (γ, β) where $\alpha \neq \gamma$. Since c_1 uniquely specifies an information word, there exist no word of the form (γ, β). Hence
$$m_{j,2}(\gamma)m_{j,1}(\beta)g_j(a) = 0$$
for all j, and the output of the checker will be $R_0 = R_1 = 0$.

■

C. Characteristic functions construction

In this subsection we present a greedy procedure for generating the characteristic function. The computational complexity of the procedure is smaller than mN^2 where m is the number of input variables and N is the number of products (cubes) that specify the functional unit.

Let $\mathcal{G} = \{0, 1, *\}$, and, $p \in \mathcal{G}$. Let a be a Boolean variable. We define a^p as

$$a^p = \begin{cases} a & if \quad p = 1 \\ \bar{a} & if \quad p = 0 \\ 1 & if \quad p = * \end{cases} .$$

Definition 5 (cube): A cube $P = (p_{m-1}, \ldots, p_0) \in \mathcal{G}^m$, of order r is a coset comprising the 2^r assignments of $X = (x_{m-1}, \ldots, x_0) \in \{0, 1\}^m$, for which the corresponding Boolean product $f_P(X) = \Pi_{i=0}^{m-1} x_i^{p_i}$ equals "1". The value of r equals to the number of $*$'s in p.

The intersection between two cubes P_i and P_j comprises the elements in the intersection of the cosets, or equivalently, the assignments of X for which $f_{P_i}(X) \cdot f_{P_j}(X) = 1$. Two cubes are called disjoint if their intersection is empty.

Let F_i denote the set of cubes $\{P_j^{(i)}\}_{j=1}^{N_i}$ that are associated with the information word Y_i. Each element X in the union of the cosets defined by F_i satisfies: $f(X) = Y_i$. Clearly, $F_i \cap F_j = \Phi$ for $i \neq j$.

TABLE I

SOP CHECKER CONSTRUCTION PROCEDURE

For each information word Y_i, $1 \leq i \leq M$,
  $A = \cup_{s|d(\hat{Y}_i,\hat{Y}_s)=1} F_s$.
  If $A = \Phi$ then $\hat{Y}_i$ is of distance two from all other words, thus,
    $\hat{F}_i = \{(*,*,\cdots *)\}$.
  Else construct the set $\hat{F}_i$:
    $\hat{F}_i = \Phi$.
    For each cube $P_j^{(i)}$ in F_i, $1 \leq j \leq N_i$,
      $P = P_j^{(i)}$.
      for $1 \leq w \leq m$,
        $tempP = P$,
        $tempP_w = *$,
        if $tempP \cap A = \Phi$, then $P = tempP$.
      end
      $\hat{F}_i = \hat{F}_i \cup P$.
    end
  end
end

Let $h_i(X)$ be the Boolean function defined by F_i, that is,

$$h_i(X) = \vee_j f_{P_j^{(i)}}(X).$$

The characteristic function $g_i(X)$ covers $h_i(X)$,

$$g_i(X) \geq h_i(X).$$

The procedure presented in Table I shows how to construct from each F_i a new set of cubes $\hat{F}_i$, that are of larger order than the original cubes. The set $\hat{F}_i$ defines a characteristic function that has a smaller number of literals than the function defined by F_i. For each information word Y_i, the procedure goes over all the cubes $P = (p_{m-1}, \ldots, p_0) = P_j^{(i)} \in F_i$. The procedure changes the symbol p_w, $0 \leq w \leq m-1$, to $*$ if after the change the modified cube P and the set $A = \cup_{s|d(\hat{Y}_i,\hat{Y}_s)=1} F_s$ remain disjoint, that is $P \cap A = \Phi$.

Example 3: The set of cubes associated with the information words in Example 1 is the following:

$$\begin{aligned} F_1 &= \{(001*0), (1*001)\}, \\ F_2 &= \{(10000), (00*11), (10110)\}, \\ F_3 &= \{(11*00)\}, \\ F_4 &= \{(00000)\}, \\ F_6 &= \{(1111*), (01000), (0*101)\}. \end{aligned}$$

All the remaining combinations are associated with the information word Y_5 :

$$F_5 = \left\{ \begin{array}{l} (011*0), (0*001), (1010*), (1*101), (10*11), \\ (**010), (01*1*), *101*) \end{array} \right\}.$$

The new set of cubes which defines the characteristic functions is:

$$\begin{aligned} \hat{F}_1 &= \{(*01*0), (**0*1)\}, \\ \hat{F}_2 &= \{(*0***)\}, \\ \hat{F}_3 &= \{(*1***)\}, \\ \hat{F}_4 &= \{(*00*0)\}, \\ \hat{F}_5 &= \{(*****)\}, \\ \hat{F}_6 &= \{(*1*1*), (*1**0), (**1*1)\}. \end{aligned}$$

Note that the the word $\hat{Y}_5$ is of distance two from all the other words. Thus, its corresponding A is empty and the cube $\hat{F}_5$ equals to $(*****)$. That is, the characteristic function associated with Y_5 is $g_5(X) = 1$.

Let F be the set that comprises all the cubes:

$$F = \cup_{i=1}^{M} F_i.$$

Denote by $|F|$ the number of products in F,

$$|F| = \sum_{i=1}^{M} N_i.$$

Let $W(P)$ be the number of literals in the product that corresponds to the cube $P = (p_{m-1}, \ldots, p_0)$,

$$W(P) = |\{w | p_w \neq *,\ 0 \leq w < m\}|.$$

We define the *density* of F as

$$\mathcal{D}(F) = \frac{\sum_{P \in F} W(P)}{m|F|}.$$

In Example 3, the density of the original set is $\mathcal{D}(F) = 73/(5*18) = 81\%$, while the density of the encoded set is $\mathcal{D}(\hat{F}) = 16/(5*9) = 36\%$. Although the *density* is not a measure of the implementation's complexity, it can be used as an indicator to the simplification that the suggested approach can provide.

IV. EXPERIMENTAL RESULTS

In this section we present results obtained from experiments with a number of industrial benchmarks provided by different high-tech companies. The results of the experiments are presented in Tables II and III.

Table II compares the suggested structure with duplication based solutions and with strict encoding (i.e. coding the information using $\lceil log_2(M) \rceil$ bits). The comparison is done in terms of the density measure. The first column of the Table contains the benchmark name; columns 2, 3, 4 contain the number of inputs m, the number of codewords M, and the number of information bits k. The number of information bits (k_1, k_2) implemented as c_1 and c_2 are written in the 5'th column. The number r of x's that are actually routed to the SOP checker are written in the 6'st column. The density of the original set $\mathcal{D}(F)$ that indicates the complexity of the duplication and the strict encoding, is given in column 7. The density of the encoded set $\mathcal{D}(\hat{F})$ is given in column 8. The last row of the table refers to normalized values of the parameters. The CPU time of the encoding procedure was measure on Intel-Centrino, 1.2Ghz, 0.99GB RAM, and is given in the 9'th column.

On average, the suggested encoding scheme improves the density by a factor of 0.54 and about 66% of the AND matrix of the PLA that specifiys the characteristic functions contains don't care values.

Table III shows the complexity of the overall system in terms of the number of Look-Up-Tables (LUTs). We used *SPARTAN3 xcs200ft256* and LeonardoSpectrum. The number of LUTs required for implementation of the functional unit as one circuit is written in the second column of the table; the number of LUTs required to implement the functional unit as two independent circuits is written in the third column.

TABLE II
DENSITY

	m	M	k	(k_1, k_2)	r	$\mathcal{D}(F)$	$\mathcal{D}(\hat{F})$	sec
s27	7	6	4	(3,1)	6	81	38	0.078
s298	17	332	20	(16,4)	13	98	44	3.078
s386	13	23	13	(11,2)	13	67	27	0.047
s420	35	36	18	(18,0)	17	55	42	0.000
s510	25	73	13	(9,4)	25	27	18	0.063
s832	23	70	24	(22,2)	22	39	30	0.563
s1494	14	168	25	(21,4)	13	64	37	0.297
total						1	0.5476	

TABLE III
NUMBER OF LUTS IN THE OVERALL SYSTEM

	single circuit	circuit c_1 and c_2	suggested scheme	ref. [11] + SOM ch.	strict enc. + SOM ch.
s27	12	(7,5)	22	25	27
s298	2410	(2312,253)	3937	5310	5733
s386	63	(56,10)	163	188	194
s420	104	(104,133)	341	430	383
s510	81	(70,11)	303	401	389
s832	346	(345,4)	725	921	1032
s1494	674	(545,135)	1322	1720	1832
total	1	1.08	1.85	2.44	2.60

Columns 4, 5, and 6 show the number of LUTs required for the implementation of the overall system, that is, the functional unit circuits plus a SOM (or SOP) checker. The 4'th column corresponds to the proposed scheme, the 5'th column corresponds to the coding scheme presented in [11] combined with a SOM checker, and the 6'th column corresponds to a system based on the strict encoding. The table clearly demonstrates the efficiency of the suggested structure. On average, the presented scheme allows detection of any arbitrary fault by increasing the implementation cost by 85%. This is better than the conventional method of duplication or strict encoding, or the method presented in [11].

V. CONCLUSIONS

The known approaches for designing independently checked circuits are all based on introducing a significant redundant portion into the original scheme. These known methods use various strategies of coding to optimize the final hardware solution according to different criteria. In our paper we avoid the necessity to introduce any additional redundancy into the initial scheme to be checked. Instead, we propose using already existing variables to achieve the required effect of detecting a single fault that may cause arbitrary errors (not necessarily unidirectional). These existing variables are the input variables of the functional circuit. In our case, some of the input variables are used as additional inputs entered into a checker, in addition to the original output variables.

We have formulated theoretical fundamentals of the proposed design method. Based on the fundamentals, we have solved a problem of selecting the optimized set of input variables to be added to the output vector.

The proposed approach has been implemented and investigated. Experimental results, obtained using a number of standard benchmarks, indicate a significant improvement in detection of arbitrary errors, in comparison with the conventional methods and in terms of the required hardware overhead.

ACKNOWLEDGMENT

The authors would like to thank Vladimir Ostrovsky and Benni Abramov for their support and assistance with this project.

REFERENCES

[1] Alidina M. et al., "Precomputation-based Sequential Logic Optimization for Low Power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp.426-436, Dec. 1994.

[2] J. M. Berger, "A Note on Error Detection Codes for Asymmetric Channels," *Information and Control*, vol. 4, pp. 68-73, Mar. 1961.

[3] C. V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Information and Control* , vol. 5, pp. 64 - 71, 1962.

[4] N.K. Jha and S.J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits," *IEEE Transaction CAD*, vol. 12, no. 6, pp. 878-887, Jun. 1993.

[5] Kaushik De., Chitra Natarajan, Devi Nair, Prithviraj Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits," *IEEE Transaction on Very Large Integration (VLSI) Systems*,vol. 2, no. 2, pp. 186-195, June 1994.

[6] P. Lala, *Self-checking and Fault-Tolerant Digital Design*, Morgan Kaufmann Publishers, San-Francisco / San-Diego / New-York/ Boston/ London/ Sydney/ Tokyo, 2000.

[7] I. Levin, M. Karpovsky, "On-line Self-Checking of Microprogram Control Units", *The 4-th IEEE International On-line Testing Workshop*, Capri, pp. 153 - 159, 1998.

[8] I. Levin and V. Sinelnikov, " Self-checking of FPGA based Control Units," *Proc. of 9th Great Lakes Symposium on VLSI*, Ann Arbor, Michigan, IEEE press, pp. 292-295, 1999.

[9] C. Metra, S. Francescantonio, M. Omana, "Automatic Modification of Sequential Circuits for Self-Checking Implementation," *Proc. of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03)*, pp.417 - 424, 2003.

[10] V. Ostrovsky and I. Levin, "Implementation of Concurrent Checking Circuits by Independent Sub-circuits," *Proceedings of 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pp. 343-351, 2005.

[11] V. Ostrovsky, I. Levin, O. Keren, B. Abramov, "Designing Concurrent Checking Circuits by using Partitioning," accepted for publication in the *International Journal of Highly Reliable Electronic System Design* on Aug-2007.

[12] I. Pomeranz and S.M. Reddy, "Recovery During Concurrent On-Line Testing of Identical Circuits," *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, pp. 475-483, Oct. 2005.

[13] V.V. Saposhnikov, A. Morosov, Vl.V. Saposhnikov, M. Gossel, "Design of Self-Checking Unidirectional Combinational Circuits with Low Area Overhead," *Proceeding of the 2nd IEEE International On-line Testing Workshop*, pp. 56-67, Jul. 1996.

[14] V.V. Saposhnikov, A. Morosov, Vl.V. Saposhnikov, M. Gossel, "A New Design Method for Self-Checking Unidirectional Combinational Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 12, pp. 41-53, Feb. 1998.

[15] J. E. Smith , "On separable unordered codes," *IEEE Transaction on Computers*, vol. 33, no. 8, pp. 741-743, Aug. 1984.

[16] E.S. Sogomonyan, "Design of Built-in Self-Checking Monitoring Circuits for Combinational Devices," *Automation and Remote Control*, vol. 35, no. 2, pp. 280-289, 1974.

[17] L. Znghel, M. Nicolaidis, I. Alzaher-Noufal, "Self-Checking circuits versus realistic faults in very deep submicron," *18th IEEE VLSI Test Symposium (VTS 2000)*, pp. 55-63, May 2000.