# Arbitrary Error Detection in Combinational Circuits by using Partitioning

Osnat Keren,
Bar-Ilan University,
*kereno@eng.biu.ac.il*

Ilya Levin,
Tel-Aviv University,
*ilia1@post.tau.ac.il*

Vladimir Ostrovsky,
Tel-Aviv University,
*vladio@post.tau.ac.il*

Beni Abramov,
Tel-Aviv University,
*abramovbeni@gmail.com*

**Abstract**

*The paper presents a new technique for designing a concurrently checking combinational circuit. The technique is based on partitioning the circuit into two independent sub-circuits. It does not require any redundant coding variables; instead, it utilizes a sub-set of input variables. These variables are transferred directly into a checker providing the arbitrary error detection. The paper develops and studies a method for selecting an optimized sub-set of such variables. Benchmark results show efficiency of the proposed approach.*

## 1 Introduction

The majority of the known concurrent checking schemes assumes that a set of output words of the functional circuit to be checked is complete, i.e., any binary vector may occur. However, it is often reasonable to construct a so-called context-oriented concurrent checking scheme, where: a) the number $M$ of possible output vectors, is much smaller than $2^k$, where $k$ is the width of the output vector, and b) the set of possible outputs is known in advance. The context-orientation has some advantages in comparison with the universality. 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 partitioning 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 [3], where a 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. Experimental results show that for $M < 2^k/3$, the system in [3] has smaller implementation cost than a solution based on duplication of the functional unit. In [4], the authors developed a specific architecture for checking sequential circuits without introducing any redundant coding variables. 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 [5] as an alternative 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, we propose a new technique that allows to detect *arbitrary* errors in a combinatorial circuit. That is, all single faults (that may cause any binary vector on the

1550-5774/08 $25.00 © 2008 IEEE
DOI 10.1109/DFT.2008.34

circuit's output) are detectable by the suggested scheme. Similarly with [4], we don't use any redundant coding for the output vectors. We propose to transfer the input variables (and not the products) into the checker. Actually, these input variables are used instead of the coding variables. A method for designing such a circuit is the main contribution of the paper. We show that the partitioning of the initial circuit into independent sub-circuits followed by choosing an optimized set of input variables is efficient for detecting both unidirectional and arbitrary errors. The proposed partitioning algorithm is heuristic and does not provide the optimal partition. Nevertheless, it is simple and provides good solutions.

The paper is organized as follows: Section 2 includes basic definitions, and recalls related work on on-line testing for arbitrary errors. Section 3 presents the suggested structure, and Section 4 contains experimental results. Section 5 concludes the paper.

## 2 Preliminaries

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 $Y_i = (y_{k-1}^{(i)}, \ldots, y_0^{(i)})$ is encoded to a codeword $Z_i = (z_{n-1}^{(i)}, \ldots, z_0^{(i)})$ 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* $z_{j_s}^{(i)} = y_s^{(i)}$ *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.

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 an arbitrary error (that is, not necessarily unidirectional), the fault may not be detected in functional units that are implemented as a single circuit. In order to detect any fault the functional unit should be implemented by at least two independent circuits [6].

Coding schemes for such a case, were discussed in [1, 6, 7, 8]. 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 output of the functional unit is denoted by $\hat{Y} = (c_2, c_1)$. Obviously, there is a one-to-one mapping between each $Y_i$ and $\hat{Y}_i$.

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 [2]. In [6] it was shown, that iff $c_2 = f_1(c_1),\ and,\ c_1 = f_2(c_2)$, then the functional unit is fault secure in respect to arbitrary errors in one of the two circuits.

There are several ways to partition the set of information bits $\{y_j\}_{j=0}^{k-1}$ into two sets. In this paper, we follow the approach introduced in [6], where partitioning is done with the aim to minimize the number of $y$'s in $c_1$ that completely specify the information words; namely,

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

The remaining information bits (if any) form $c_2$. Note that this partition may not be the optimal in terms of the implementation cost.

**Definition 2 (Distance)** *The distance $d(\hat{Y}_i, \hat{Y}_j)$ between two words $\hat{Y}_i = (c_2^{(i)}, c_1^{(i)})$ and $\hat{Y}_j = (c_2^{(j)}, c_1^{(j)})$ is $d(\hat{Y}_i, \hat{Y}_j) = |\{k | c_k^{(i)} \neq c_k^{(j)}),\ k = 1, 2\}|$.*

**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 one of the circuits implementing $c_1$ or $c_2$ is detectable if and only if all codewords are of distance two from each other. Nevertheless, in some cases, it is impossible to find a partition that leads to a $d_{min}$ that equals two ($d_{min} = 2$). The following example illustrates such a case.

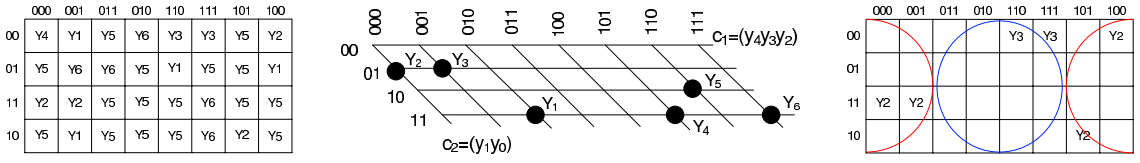
**Example 1** *Consider a functional unit that has five inputs $X = (x_4, \ldots x_0)$ and five outputs $Y = (y_4, \ldots y_0)$. Assume that the system produces only $M = 6$ information words $\{Y_i\}_{i=1}^{6}$ out of the possible $2^5$ combinations:*

$$Y_1 = (01011),\ Y_2 = (00001),\ Y_3 = (00101),\ Y_4 = (10111),\ Y_5 = (11010),\ Y_6 = (11111).$$

*The functionality of the unit is given in Figure 1(a) in a Karnaugh-like map.*

*Let $c_1 = (y_4, y_3, y_2)$ and $c_2 = (y_1, y_0)$. The location of the information words as points in the $c_1 \times c_2$ plane is shown in Figure 1(b). A fault in $c_1$ (or $c_2$) is equivalent to moving a word to an arbitrary position on the $c_1$ (or $c_2$) axis without changing its position on the other axis. It is clear from the figure, that each word has its own location on the $c_1$ axis. Therefore, a fault in $c_2$ cannot shift one information word onto another. This is not the case with a fault in $c_1$; A fault in $c_1$ may shift the point corresponding to $Y_6$ onto the point representing $Y_1$.*

*The distance between $\hat{Y}_1$ and $\hat{Y}_2$ is $d((010, 11), (000, 01)) = 2$. However, 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 equals one. 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 impossible to detect a fault in the circuit that implements $c_1$ unless 'side-information' is given to the checker.*




(a) K-map (b) The $c_1 \times c_2$ plane (c) K-map for $Y_2$ and $Y_3$

Figure 1: K-map for the functional unit in Ex. 1 (left), K-map for the characteristic functions of $Y_2$ and $Y_3$ in Ex. 2 (center), and the location of the information words on the $c_1 \times c_2$ plane (right).

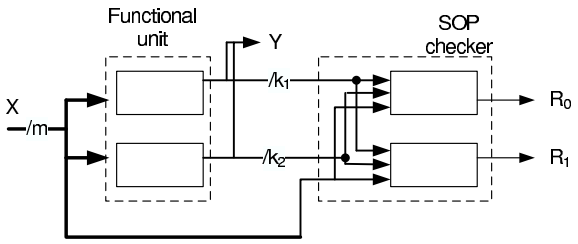



Figure 2: Proposed architecture of a non redundant functional unit with SOP checker

# 3 Arbitrary error detecting architecture

## 3.1 General error detection architecture

In this paper, we suggest a new approach for detecting an arbitrary single fault in the functional unit. Instead of encoding an information word $Y_i$ into a codeword $Z_i$ by adding $r$ redundancy bits, we suggest to use the (output) information bits as they are (uncoded), and use the $x$'s as additional 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 architecture is shown in Figure 2. 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 input variables $X$ together with $\hat{Y} = (c_2, c_1)$ enter a (Sum-Of-Products) SOP based checker. While the conventional SOM checker cannot be minimized (since it comprises minterms of distance two), the suggested checker comprises: a) products and not minterms, and, b) this products may further minimized.

## 3.2 Characteristic functions and SOP checker

In this subsection we present a checker that is based on products rather than 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. 1. The inputs to the checker are $X$ and the output $\hat{Y}$ of the functional unit.

Recall that the input lines are routed directly to the checker. The inputs are used to provide 'side-information' on the 'problematic', that is, information words that are of distance one from each other. In this paper we consider two cost functions:

- *Minimal number of inputs.* That is, use of the minimal number of $x$'s that are sufficient to provide desired property.
- *Minimal density.* That is, use of a set of $x$'s that reduce the number of literals in the SOP representation of the checker's function. In other words, the complexity criteria is the implementation cost of the checker (e.g. two-input AND-OR gates).

We start by defining a Characteristic function in respect to a given partition:

**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*

$$g_i(X) = \begin{cases} 1 & f(X) = Y_i \\ 0 & f(X) = Y_j \text{ and } d(\hat{Y}_i, \hat{Y}_j) = 1 \\ \phi & \text{otherwise} \end{cases} \quad (2)$$

*where $\phi$ stands for don't care.*

Note that the characteristic functions are not necessarily pairwise orthogonal (disjoint). That is, $\sum_X g_i(X)g_j(X) \geq 0$.

**Example 2** *The codeword $\hat{Y}_2$ in Example 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 1(c) 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. A characteristic function for $g_2(X)$ may be $g_2(x) = x'_4x'_3x_1x_0 + x_4x'_3$. Clearly, the simplest expressions for the characteristic functions are $g_2(X) = x'_3$ and $g_3(X) = x_3$.*

A SOP checker is based on dividing the set $\mathcal{Y}$ into two non-empty and disjoint sets $\Pi_0$ and $\Pi_1$. The SOP checker consists of two independent circuits. Each circuit implements the function

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

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

### 3.3 Characteristic functions construction

In this subsection we present two greedy approaches for generating the characteristic function in respect to the complexity criteria mentioned above.

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 $p$.*

An intersection between two cubes $P_i$ and $P_j$ comprises 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 $F_i$ be 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 $i \neq j$.

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

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

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

$$F_5 = \{\ (011*0), (0*001), (1010*), (1*101), (10*11), (**010), (01*1*), (*101*)\ \}.$$

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

### 3.4 Characteristic functions minimizing the number of literals

In order to reduce the number of literals, each original set of cubes $F_i$ has to be covered by a new set of cubes $\hat{F}_i$, that are of larger order than the original cubes. Namely, let $P = (p_{m-1}, \ldots, p_0) = P_j^{(i)} \in F_i$ be a cube in the set associated with the information word $Y_i$. Changing one symbol of $P$ from 0 (or 1) to $*$ defines a cube $\hat{P}$ that has larger order and covers $P$. The $w$'th symbol of $P$, symbol $p_w$, $0 \leq w \leq m-1$, can be changed to $*$ if after the change the modified cube $\hat{P}$ and the set $A_i = \cup_{s|d(\hat{Y}_i,\hat{Y}_s)=1} F_s$ remain disjoint, that is $\hat{P} \cap A = \Phi$.

**Example 4** *The set $F_1$ consists of two cubes, $P_1 = (001*0)$ and $P_2 = (1*001)$. The corresponding set $A_1$ is $A_1 = F_4 \cup F_6$. Clearly, $P_1 \cap A_1 = \Phi$. However, it is possible to change the 5'th symbols of $P_1$ to $*$ and still have disjointness, that is, $(*01*0) \cap A_1 = \Phi$. Similarly, the 2'nd and 5'th symbols of $P_2$ can be changed to $*$ while keeping the disjointness property. Therefore, the set $\hat{F}_1 = \{(*01*0), (**0*1)\}$ covers $F_1$ and forms a characteristic function. A new set of cubes which defines characteristic functions that have less literals is:*

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

*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.

### 3.5 Characteristic functions with minimized number of inputs

Let $h_i(X)$ be the Boolean function defined by a set of cubes $F_i$, that is associated with the information word $Y_i$ : $h_i(X) = \vee_j f_{P_j^{(i)}}(X)$. By its definition, $h_i(X)$ is a characteristic function of $Y_i$. Denote by $\hat{X}$ a subset of the input variables. Let $\hat{F}_i$ be a set of cubes that is constructed from $F_i$ by assigning $*$ at positions that correspond to $x$'s that are not in $\hat{X}$, that is, $\hat{F}_i = \{\hat{P}_j^{(i)}\}_{j=1}^{N_i}$ where $\hat{P} = (\hat{p}_{m-1}, \ldots, \hat{p}_0) \in \mathcal{G}^m$, and

$$\hat{p}_i = \begin{cases} p_i & x_i \in \hat{X} \\ * & x_i \notin \hat{X} \end{cases}.$$

Clearly the Boolean function $\hat{h}_i(\hat{X})$ that corresponds to $\hat{F}_i$ covers $h_i(X)$, that is, $\hat{h}_i(\hat{X}) \geq h_i(X)$. Nevertheless, $\hat{h}_i(\hat{X})$ may not be a characteristic function.

**Example 5** *Consider the function in Example 3. For* $\hat{X} = \{x_1, x_0\}$, *we have,* $\hat{F}_3 = \{(***00)\}$. *The corresponding function,* $\hat{h}_3(\hat{X}) = x_1'x_0'$, *is not a characteristic function. However, for* $\hat{X} = \{x_3, x_0\}$, *the function* $\hat{h}_3(\hat{X}) = x_3x_0'$ *is a characteristic function.*

Denote by $\hat{X}_{opt}$ the the minimal subset of $x$'s for which all the corresponding functions $\{\hat{h}_i(\hat{X}_{opt})\}_{i=1}^{M}$ are characteristic functions. Then, the functional unit combined with a SOP checker that uses these characteristic functions is fault secure.

**Example 6** *The minimal set of inputs for the function in Example 3 is* $\hat{X}_{opt} = \{x_3, x_2, x_0\}$ *and the corresponding set of products is:*

$$\begin{array}{lcllcl}
\hat{F}_1 & = & \{(*01*0), (**0*1)\}, & \hat{F}_2 & = & \{(*00*0), (*0**1), (*01*0)\}, \\
\hat{F}_3 & = & \{(*1**0)\}, & \hat{F}_4 & = & \{(*00*0)\}, \\
\hat{F}_5 & = & \left\{ \begin{array}{l} (**0*1), (*01**), (**1*1), (*0**1), \\ (**0*0), (*1***) \end{array} \right\}, & \hat{F}_6 & = & \{(*11**), (*10*0), (**1*1)\}.
\end{array}$$

Clearly, the two approaches for reducing the checker's complexity, reducing the number of inputs and reducing the density, can be combined.

## 4 Experimental results

In this section we present results obtained from experiments with a number of ISCAS89 benchmarks; we used the combinatorial part of these sequential circuits which inherently have the context orientation property. The results of the experiments are presented in Tables 1 and 2.

Table 1 compares the suggested structures with duplication 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, $k_1$ and $k_2$, which are implemented as $c_1$ and $c_2$, are written in the 5'th column. The number $r_d$ of $x$'s in a checker having minimal number of literals (minimal density) is is given in column 6, and the minimal number $r_i$ of $x$'s that can be used to construct the checker is given in column 7. The density of the original set $\mathcal{D}(F)$ that indicates the complexity of the duplication and the strict encoding, and the density of the encoded sets are given in columns 8 to 11. The density of a checker designed for minimal number of literals and the density of a checkers designed to minimal number of inputs are denoted by $\mathcal{D}(\hat{F}_d)$ and $\mathcal{D}(\hat{F}_i)$, respectively. The density of a checker that combines both properties is denoted by $\mathcal{D}(\hat{F}_c)$. The last row of the table refers to normalized values of the parameters. The CPU-time of both encoding procedures is given in the two last columns.

Notice that in some cases (e.g. *s420*), no partitioning is possible; in such cases $k_2 = 0$.

On average, the encoding scheme that is based on minimizing the number of literals, 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.

The simulation results show that the number of inputs $r_i$ cannot be reduces significantly in respect to $r_d$. The overall improvement in the density obtained by combining the two

Table 1: Density and CPU-time

| | $m$ | $M$ | $k$ | $(k_1, k_2)$ | $r_d$ | $r_i$ | $\mathcal{D}(F)$ | $\mathcal{D}(\hat{F}_d)$ | $\mathcal{D}(\hat{F}_i)$ | $\mathcal{D}(\hat{F}_c)$ | $sec_d$ | $sec_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s27 | 7 | 6 | 4 | (3,1) | 6 | 4 | 81 | 38 | 57 | 34 | 0.078 | 0.141 |
| s298 | 17 | 332 | 20 | (16,4) | 13 | 13 | 98 | 44 | 76 | 44 | 3.078 | 75.484 |
| s386 | 13 | 23 | 13 | (11,2) | 13 | 12 | 67 | 27 | 92 | 26 | 0.047 | 3.312 |
| s420 | 35 | 36 | 18 | (18,0) | 17 | 17 | 55 | 42 | 42 | 42 | 0.000 | 0.031 |
| s510 | 25 | 73 | 13 | (9,4) | 25 | - | 27 | 18 | - | - | 0.063 | $> 5E+4$ |
| s832 | 23 | 70 | 24 | (22,2) | 22 | 22 | 39 | 30 | 59 | 30 | 0.563 | 1.192 E+4 |
| s1494 | 14 | 168 | 25 | (21,4) | 13 | 13 | 64 | 37 | 93 | 37 | 0.297 | 15.39 |
| average | | | | | | | 1 | 0.5476 | 1.0371 | 0.5272 | | |

Table 2: Number of LUTs in the overall system

| | single circuit | circuit $c_1$ and $c_2$ | suggested scheme | ref. [6] + 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 |

approaches, is negligible in respect to the density of the checker designed for minimal number of literals.

Note, that the complexity (in terms of computation time) of constructing a checker that has minimal number of inputs is much larger than the complexity of obtaining a checker with minimal number of literals. In light of the small difference between the overall densities, we find the large computational time consumed for minimizing the number of inputs, unjustified.

Table 2 shows the complexity of the overall system in terms of the number of Look-Up-Tables (*LUT*s). 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. Columns 4, 5, and 6 show the number of LUTs required for implementing the overall system, that is, the functional unit circuits plus a SOM (or SOP) checker: the 4'th column corresponds to the proposed scheme with reduced density, the 5'th column corresponds to the coding scheme presented in [6] 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 [6].

## 5 Conclusions

Known techniques for designing concurrently checking circuits are usually based on introducing a significant redundant portion into the original scheme. Reducing this redundant portion is one the main concerns in designing concurrently checking circuits. In a case when all possible circuit's output vectors are known in advance the designing of the such circuits may be simplified by using so-called context-oriented techniques. One of the

context-oriented techniques that is in the focus of our study is based on the partitioning of the initial circuit for two independent sub-circuits. The partitioning utilizes the context-orientation property of the original circuit by using correlation between its output variables.

In our paper, we proposed a technique that being based on the partitioning avoids the necessity to introduce any additional redundancy into the initial scheme to be checked. It uses already existing input variables to achieve the required effect of detecting an arbitrary fault. Some of the input variables are used as additional inputs entered into a checker, in addition to the original output variables of the circuit to be checked.

We have formulated theoretical fundamentals of the proposed design method. Based on the fundamentals, we proposed a solution of the 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.

## References


[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.