Designing Concurrent Checking Circuits by using Partitioning[1]

Vladimir Ostrovsky*, Ilya Levin*, Osnat Keren**, and Binyamin Abramov*

Tel Aviv University, Bar Ilan University**, Israel*

Abstract

The paper describes a new method for synthesis of concurrent checking logic circuits having a limited number of codewords. This method is based on implementing such a circuit by a network of separate independent subcircuits. Each of these subcircuits implements its own subset of output variables. Since the subcircuits have no common elements, any single fault results in errors only in one of the subsets. The partition of the circuit is made in such a way that the subcircuits are able to check each other. Two ways of selecting the optimal partition are examined: a) partitioning that minimizes a number of subcircuits; and b) partitioning that minimizes a number of encoded variables. It is shown that a trade-off between these two ways allows one to select the optimal solution for the synthesis. The benchmarks' results indicate the efficiency of the proposed technique.

1. 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 [1]. The failures phenomena, together with the need for higher reliability of complex digital systems, are of special interest. As the microelectronics industry moves towards 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 [2].

Numerous publications concerning designing on-line checking circuits have presented significant results, but the problem remains unresolved. The problem of developing highly reliable and economical hardware methods for detecting faults (errors) remains a real and important task in practice.

Our work presents a new way of designing a concurrently checking functional circuit with the aid of partitioning the circuit into a number of independent sub-circuits. For the sake of simplicity, we consider only combinational circuits. Faults (both temporary and permanent) may occur in the functional circuit and may lead to a distortion of one or more positions of an output codeword. Traditionally, it is assumed that the time interval between any two faults is sufficient for eliminating the

[1] This research was supported by the Israeli Science Foundation under grant No. 545/04.

consequences of the first fault before the second fault occurs. Therefore, only one fault may occur in a circuit at a given time. This assumption is used for constructing models of acceptable distortions of output words [3].

The majority of the known concurrent checking schemes assume 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, M is much smaller than 2^m, where m is the width of the output vector ($M << 2^m$) 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 subcircuits 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 idea of using the context orientation for partitioning a functional circuit for mutually checking components was studied by [4]. In that work, the authors examined only a two-block partition, thus 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 concurrent checking is studied more in-depth. It is shown that the partitioning may be performed in different ways and in accordance with a number of optimization criteria. We show that the partitioning into independent sub-circuits is efficient for detecting both unidirectional and arbitrary errors.

The partition-based approach allows different trade-offs to be handled. One of them is a trade-off between the number of subcircuits and a number of additional check variables that have to be introduced into the circuit. We examine the following two tasks: a) minimizing a number of coding variables while the circuit is divided into two sub-circuits; b) minimizing the number of sub-circuits while just one coding variable is introduced.

The paper is organized as follows. Section 2 presents some important definitions and a brief overview of related error-detecting techniques. In Section 3, a new detecting partition concept is introduced and a number of its properties are described and proven. Algorithms of optimized partitioning are presented in Section 4. Experimental results are presented in Section 5. Conclusions are provided in Section 6.

2. Definitions and Related Works

Let $Y = \{y_1, ..., y_m\}$ be a set of output variables of the functional circuit to be checked. We call these variables *functional* variables. Let $c_1, \ldots, c_{n-m}$ be a set of $(n-m)$ check variables that are added to the set of functional variables. Let $Z = \{z_1, \ldots, z_n\}$ be the common set of encoded variables, including those being checked and the additional ones,

$$z_i = \begin{cases} y_i, & if \quad i \le m; \\ c_{i-m}, & if \quad i > m. \end{cases}.$$

During a normal operation of the circuit, the variables z_i take values from a set of codewords $A = \{a_1, \ldots, a_M\}$. The set of codewords is *complete* if $M = 2^m$ and *incomplete* iff $M < 2^m$. The complete set of m-bit words is denoted by W_m.

To distinguish the encoded words from the non-encoded (original) words, we denote the non-encoded words and the corresponding sets of the words by the symbol "-" as follows: $A^- = \{a_1^-, \ldots, a_M^-\}$. Note that a_i is an n–dimensional vector whereas a_i^- is an m - dimensional vector.

We define a fault model by operator ψ as follows.

Let a codeword $a_i \in A$ take an erroneous value from a set $\psi(a_i)$ because of a fault in the circuit. Let $\psi(A) = \bigcup_{a_i \in A} \psi(a_i)$. If a functional circuit satisfies the fault-secure property i.e., iff:

$$\psi(A) \cap A = \varnothing, \qquad (1)$$

then the set of words A allows detecting erroneous values designated by the operator ψ.

Let a function $R = R(a)$ be a *checking* function on the set of variables Z. That is, the function R allows distinguishing a codeword from a non-codeword: $R(a) = 1$ if $a \in A$, and $R(a) = 0$ if $a \in \psi(A)$. The function takes arbitrary values for vectors that are not in $\psi(A) \cup A$.

If a fault model on the set $\psi(A)$ specifies some restrictions, then Eq. (1) can be fulfilled by using suitable error-detecting methods. The majority of relevant publications use either the following two models of possible distortions in codewords or some combinations thereof.

The first model is based on the assumption that the system of logic functions describing a functional circuit is monotonic. In such circuits, any single fault may lead only to unidirectional errors in the output codeword. The most popular code for detecting unidirectional errors is the universal Berger code [5]. The context-oriented Smith code [6] and a separable $m-out-of-n$ code [7] are well known. A method for synthesizing inverter-free circuits is described in [8]. Another solution that is based on duplication of some elements of the circuit is proposed in [9], where a method for the design of unidirectional combinational circuits is presented.

The second model assumes that the number of distorted bits in the word cannot exceed a predetermined threshold t [3]. Such a model can be applied to circuits, where the splitting coefficient of the input gates is relatively small and each of the gates participates in forming a relatively small number of output signals. The paper [10], which is based on a study of numerous benchmarks, shows that usually a single fault results in errors in two or less bits of a codeword $(t \le 2)$.

In order to detect faults in a scheme satisfying both of the above models, one uses the Bose-Lin codes [11]. The paper [12] describes an algorithm that allows determining the

maximal value of t for an arbitrary scheme, and therefore allows simplifying the checking scheme so that the overhead can be lowered by up to 25%. There are circuits where using the above models is insufficient, since Eq. (1) is not satisfied. Here a duplication-based solution should be considered.

In this paper, in all cases (excluding Theorem 2 case), we assume that there are no restrictions regarding the operator ψ. Let us call such an operator *universal*. This means that if the circuit is not partitioned into subcircuits, then any word from the set W_s may appear on the circuit's output instead of the correct s -bit codeword b, namely, $\psi(b) = W_s \setminus b$. Thus, there is no encoding that would ensure fulfillment of Eq. (1), and therefore there is no code that can distinguish between a codeword and a non-codeword. As was already mentioned, in such cases, the checking is usually based on the duplication.

It is important to emphasize that duplication allows detecting any error resulting in a single fault in a circuit because the two identical circuits of the duplicated scheme have no common elements and not because a large number of additional variables are used. Similarly, the proposed partition of the circuit into independent sub-circuits restricts affects a single fault on the output variables and consequently restricts the set $\psi(A)$, fulfilling Eq. (1).

The following example illustrates the above idea.

Example 1. Let y_1, y_2, y_3, y_4 be output variables and let $A = \{0100, 1001\}$ be a set of two codewords. If the error operator ψ is universal, then $\psi(A) = W_4$ and, consequently, Eq. (1) is not satisfied.

Note that in this case, there exists no encoding that can fulfill Eq. (1). Let us divide the circuit into two separate subcircuits. The first subcircuit implements $Z_1 = \{y_1, y_2\}$ and the second subcircuit implements $Z_2 = \{y_3, y_4\}$. To emphasize the fact that the circuit has been partitioned, we use $\hat{\psi}$ instead of ψ. Now any fault may affect Z_1 or Z_2. In this case, there are no restrictions on the fault model at each subcircuit, i.e., outputs of the subcircuits may have any values from $W_2 = \{00, 01, 10, 11\}$. For instance,

$$\hat{\psi}(0100) = \{0000, 1000, 1100, 0101, 0110, 0111\}$$

whereas

$$\hat{\psi}(A) = \{0000, 1000, 1100, 0101, 0110, 0111, 0001, 1101, 1010, 1011\}.$$

In this case, $\hat{\psi}(A) \bigcap A = \varnothing$, consequently, all the codewords may be separated from the non-codewords.

Other solutions are usually based on redundant encoding combined with independent implementation of functional variables and do not use duplications. More economic solutions are obtained by using a method of parity checking [13-16] and by sorting the set of functional variables into groups. Variables of each group are pair-wise independent. Because of this, any single fault may cause only one error in each group. An error can be detected by adding a single check variable to each group. Paper [14] proposes a method for optimally grouping the variables and an efficient design procedure based on a local modification of the given circuit. The approach

proposed in [15] uses independence between groups. Each of the groups is implemented by a separate sub-circuit. The authors use variation of the above parity checking: one variable is taken from each group and these variables form a specific subset. A single check variable is added and used within the subset. All the variables in the subset are compared by parity. Based on the benchmarks' results, the authors of [15] concluded that the optimal partition for the considered case is a partition where each variable under check forms a separate group, i.e., is implemented by a separate independent scheme. A method proposed in [16] uses partitioning of a circuit into a number of cascades implemented by independent schemes. Outputs of the schemes are checked by parity.

In our paper, a new partitioning-based approach is developed. More specifically, the functional circuit is divided into a number of independent components (subcircuits). In contrast with the above-mentioned works, in the proposed approach the additional check bits are not parity bits. The additional bits are, in most cases, *a nonlinear* function of the functional bits. This allows more efficient encoding.

The newly introduced approach opens up new possibilities for the concurrent checking. It enables both decreasing the number of subcircuits for a constant number of check bits, and reducing the number of check bits for a constant number of subcircuits.

3. Partitions on a Set of Functional Variables

The present section focuses on the fundamentals of constructing error-detection partitions on a set of functional variables. The first subsection (3.1) introduces the concepts of a detection partition and describes its properties. Sections 3.2 and 3.3 are devoted to partitioning for detection of arbitrary errors. Section 3.4 focuses on partitioning for detection of unidirectional errors.

3.1. Partitioning for detection of arbitrary errors

Let $P=\{Z_1,\ldots,Z_k\}$ be a partition on the set Z, which means that Z is partitioned into k blocks $Z_1,\ldots,Z_k$. Denote the number of variables in Z_i by n_i, $\sum_{i=1}^{k} n_i = n$. We state that variables of block Z_i form the i-th *field* of the codeword.

Assume that each of blocks Z_i of the partition $P=\{Z_1,\ldots,Z_k\}$ corresponds to a certain independent circuit. Two circuits are considered *independent* if they do not comprise common logical elements. A single fault may occur only in one of the independent sub-circuits and, consequently, only the variables belonging to one of the blocks can be simultaneously erroneous.

By analogy with the codes, we call partition $P=\{Z_1,\ldots,Z_k\}$ a *detecting partition*, i.e., a partition that can detect the errors defined by function ψ, if and only if the set $\psi(A)$ of erroneous words meets Eq. (1).

Let B_i be a set of assignments of variables of block Z_i, $i\in\{1,\ldots,k\}$ in the codewords, $B_i\subseteq\{0,1\}^{n_i}$. The codewords determine a relation of *compatibility* between elements of any two sets B_i and B_j.

We call elements $b_i \in B_i$ and $b_j \in B_j$ $i, j \in \{1,\ldots,k\}$ *compatible* if both of these values occur in at least one of the codewords. In some specific cases, the compatibility relation may be a function. If $b_i \in B_i$ is a function of $b_j \in B_j$, this means that the values of the variables of the set Z_i, in all the codewords, are uniquely defined by the variables of the set Z_j. If a relation of this type exists, then we write $Z_i = F(Z_j)$.

In Example 1, $P = \{Z_1, Z_2\}$, where $Z_1 = \{y_1, y_2\}$ and $Z_2 = \{y_3, y_4\}$. Since the values of variables from one block define the values of variables from the other block, then: $Z_1 = F(Z_2), Z_2 = F(Z_1)$ or $Z_1 = F(Z \setminus Z_1), Z_2 = F(Z \setminus Z_2)$, which is the same.

Theorem 1. Partition $P = \{Z_1,\ldots,Z_k\}$ is the detecting partition, if and only if for each block, Z_i, the following equation is correct:

$$Z_i = F(Z \setminus Z_i) \qquad (2)$$

Proof. Let Eq. (2) be fulfilled, and let (because of a fault in the circuit) an erroneous vector $\hat{b}$ be formed instead of the correct vector $b \in B_i$. Denote by $\varphi_A(b_i = b)$ sets of codewords, the i-th field of which has values of b_i $i \in \{1,\ldots,k\}$. Then, either $\hat{b}_i \notin B_i$ and $\varphi_A(b_i = \hat{b}) = \varnothing$, or (as a result of Eq. (2)) $\varphi_A(b_i = b) \cap \varphi_A(b_i = \hat{b}) = \varnothing$. In both cases, Eq. (1) is satisfied and, consequently, any erroneous word differs from a codeword.

To prove that Eq. (2) is necessary, let us assume that a block Z_i exists, for which Eq. (2) is not fulfilled. This means that one can find at least one vector in $Z \setminus Z_i$ that is compatible with at least two different vectors $b, \hat{b} \in B_i$. Since $\hat{b} \in \psi(b)$, Eq. (1) is not satisfied and such an error cannot be detected. Thus, the theorem is proven.

Example 2.

Let us illustrate Theorem 1 by using the example of codewords shown in Table 1. In this table $z_1,\ldots,z_6$ are functional variables and z_7 is a single check variable, $z_7 = \overline{z_1 \oplus \cdots \oplus z_6}$.

Table 1. Set of codewords for Example 2.

	y_1 z_1	y_2 z_2	y_3 z_3	y_4 z_4	y_5 z_5	y_6 z_6	c_1 z_7
a_1	1	1	0	1	0	1	1
a_2	1	0	1	1	0	0	0
a_3	0	1	0	0	1	0	1
a_4	1	1	1	1	0	0	1
a_5	1	0	1	0	1	0	0
a_6	0	1	1	1	0	0	0
a_7	1	0	0	0	0	1	1
a_8	1	1	0	0	1	1	1

A codeword can be represented as a point in a three-dimensional space. Consider the partition $P_1 = \{Z_1, Z_2, Z_3\} = \{\overline{z_1 z_3 z_5}, \overline{z_2}, \overline{z_4 z_6 z_7}\}$. The corresponding partition is shown in Figure 1. The coordinates of $a \in A$ are $a = (b_1, b_2, b_3)$, where $b_i \in B_i, i = 1,2,3$. For example, the codeword a_7 is placed at $b_1 = 100, b_2 = 0$ and $b_3 = 011$. A fault in a subcircuit moves a point (codeword) along the corresponding axis. It is clear from the figure that two points cannot coincide owing to a single fault. Therefore, any error is detectable.

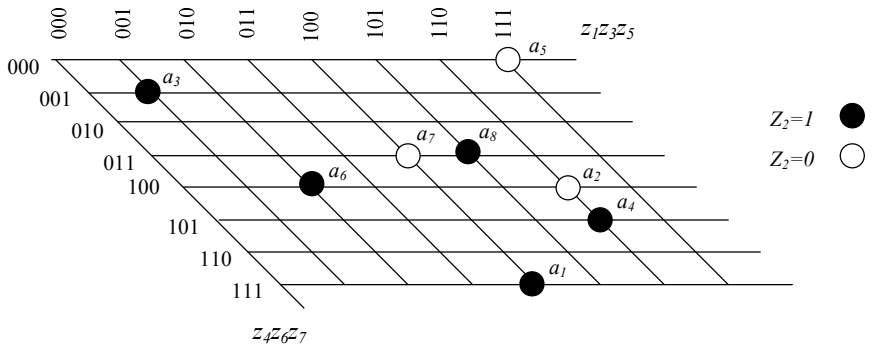

Figure 1. Three-dimensional representation of partition P_1

One can readily see that this partition allows detecting any arbitrary error. Indeed, Eq. (2) is satisfied. For example, if a fault occurs in a subcircuit implementing z_2, the output codeword $z_1, \ldots, z_7$ takes one of the erroneous values V_2= {1001011, 1111000, 0000101, 1011001, 1110100, 0011000, 1100011, 1000111} and $V_2 \bigcap A = \varnothing$.

Similarly, it is clear that if a fault occurs in one of the other subcircuits, the corresponding sets of erroneous words V_1 and V_3 also do not include codewords. Variables belonging to any two blocks of the partition P_1 determine the codeword uniquely, and consequently, they define the values of a third block of variables.

Consider, for example, another partition $P_2 = \{Z_4, Z_3\}$, where $Z_4 = Z_1 \bigcup Z_2$. Figure 2 shows the location of the codewords as points in a two-dimensional space (note that the codewords a_2 and a_6 differ only in one coordinate). For this partition, Eq. (2) is not satisfied. This partition does not allow detecting errors. As a result, a fault in block Z_4 may lead to the appearance of codeword a_6 instead of codeword a_2 in the output, and there is no indication that this situation is erroneous.

It is clear from the example that the main problem is to define a minimal number of partitions k and/or to add a minimal number of check variables c_i, such that the points (codewords) in the k-dimensional space will differ by at least two coordinates.

Consequence 1 from Theorem 1 formulates the problem of defining a detecting partition in terms of so-called differences between codewords.

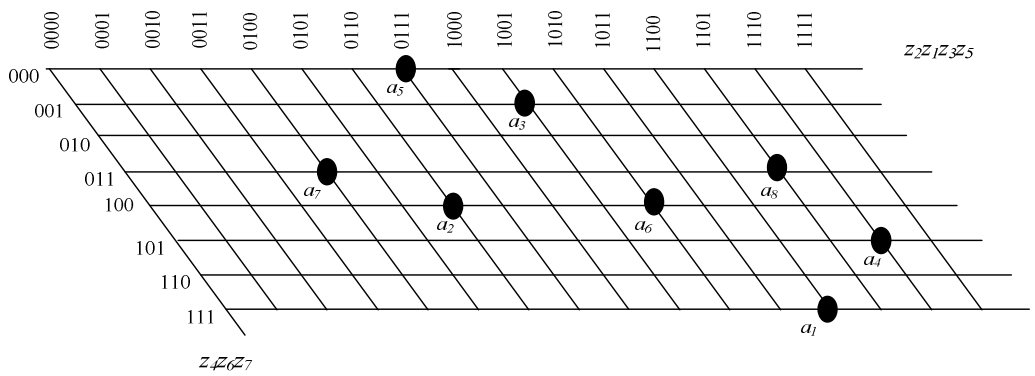

Figure 2. Two-dimensional representation of partition P_2

3.2. Partitioning differences of codewords

Let us define a *difference* $\delta(a_i, a_j)$ between any two codewords a_i and a_j as a set of variables that have different values in these words. For example, the difference between codewords a_2=101100 and a_6=0111000 equals $\delta(a_2, a_6) = \{z_1, z_2\}$.

Consequence 1. For providing to the partitioning $P = \{Z_1, \ldots, Z_k\}$ an ability to detect any error in words from the set $A = \{a_1, \ldots, a_M\}$, it is necessary and sufficient that for each two words a_i and a_j from A the following condition be met:

$$\delta(a_i, a_j) \not\subset Z_l, i, j \in \{1, \ldots, M\},\ l \in \{1, \ldots, k\}. \quad (3)$$

In other words, the difference between any two codewords cannot belong to only one block of those codewords obtained by the partition.

Proof. To prove the necessity of Eq. (3), we assume that this condition is not satisfied. This means that there are two words, a_i and a_j, having the difference that belongs to one and the same block of partition Z_l. Hence, at least two different combinations of values of variables from the block correspond to equal values of variables not belonging to Z_l, which means that $Z_l \neq F(Z \setminus Z_l)$. To prove the sufficiency of Eq. (3), we assume that Eq. (3) is satisfied but that Eq. (2) is not satisfied. This means that the partition contains such a block Z_l that $Z_l \neq F(Z \setminus Z_l)$. It is possible that when two words reflect the difference, they belong to the same block where Z_l exist. This contradiction concludes the proof.

Note that if the difference between two codewords comprises exactly two variables, in any detecting partitioning these two variables must belong to different blocks of the partition (according to Eq. (3)). If A^- comprises at least one pair of words differing, one from another, by a value of only one binary variable, then no detecting partition exists for this set A^-. In this case, before constructing the detecting partition, at least one coding variable should be added to the set, in order to obtain the minimal distance between no less than two words.

This conclusion is well supported by the known results of the coding theory. However, in the general case, partitioning of a circuit into separately implemented

sub-circuits opens up a new way for detecting errors and thus changes the coding requirements. For example, for detecting errors in not more than t bits, the requirement of the minimal distance $t+1$ between the words becomes not obligatory.

In Example 2, illustrated in Table 1, the words are encoded by a single additional variable z_7 in such a manner that any one of the differences comprises at least two variables. The differences between the words are presented in the shape of a triangle in Table 2.

Table 2. Difference in the relations regarding the set of codewords

a_2	0110011						
a_3	1001110	1111101					
a_4	**0010010**	**0100001**	1011100				
a_5	0111111	**0001100**	1110001	0101101			
a_6	1010011	**1100000**	0011101	**1000001**	1101100		
a_7	**0101000**	0011011	1100110	0111010	0010111	1111011	
a_8	**0001100**	0111111	**1000010**	0011110	0110011	1011111	**0100100**
	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Any element $\delta_{i,j}$ of Table 2 is a binary vector, of which non-zero components correspond to the variables that belong to $\delta\left(a_i, a_j\right)$; $\delta_{i,j} = a_i \oplus a_j$. It is clear that all differences comprising more than two elements can be removed if each of such differences comprises at least one two-element difference. The bold font in the table denotes elements that remain upon removing the absorbing ones. Thus, in the discussed example, Consequence 1 states a kind of "incompatibility" between variables, namely, incompatible variables cannot belong to one and the same block of the partition. Table 3 visually demonstrates such incompatibility: the incompatible pairs of variables are denoted by "1". Each partitioning block is a sub-set of compatible variables.

Table 3. Incompatibility relations

z_2	1					
z_3	0	0				
z_4	0	**1**	0			
z_5	0	**1**	0	1		
z_6	**1**	0	1	0	0	
z_7	**1**	**1**	0	0	0	0
	z_1	z_2	z_3	z_4	z_5	z_6

Let us represent the above relations by a *compatibility graph.* The compatibility graph for our example is shown in Figure 3, where two variables are connected by an edge if they are compatible. The variables of a partitioning block form a complete sub-graph. In the discussed example, the variables z_1, z_2, z_7 are pair wise incompatible. Therefore, the number of blocks in any fault that detects partitioning is three or more. One of the two possible triple-block partitioning P_1 was discussed earlier. However, if we assume that only unidirectional faults may appear in the scheme, the number of partitioning blocks can be reduced to two (see Section 3.4).

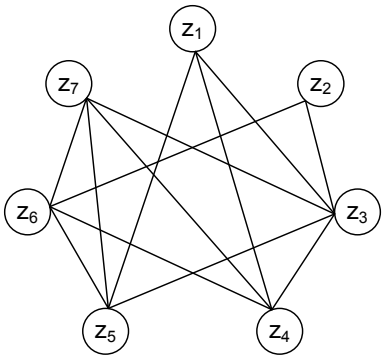

Figure 3. Compatability relation graph for Example 2.

3.3. Partitions on the set of codewords

In some cases, a so-called *algebra of partitions* [17] can be used for obtaining the optimal partitioning. Partitioning the set of output variables into blocks determines another type of partitioning that is also important in the proposed approach, namely, partitioning on the set of codewords A. Whereas for partitions on the set of output variables we use the letter P, the second type of partitions on the set of codewords is denoted by the letter π. The relation between the two types of partitions is as follows:

Each block $Z_i \left(1 \le i \le k\right)$ of the partition $P = \left\{Z_1, \ldots, Z_k\right\}$ determines a partition π_i on the set A of codewords. A certain block of π_i comprises all the codewords having the same values of variables from set Z_i. In our example of the partition, $P_1 = \left\{Z_1, Z_2, Z_3\right\}$ and $P_2 = \{Z_4, Z_3\}$ determine the following partitions on the set of codewords:

$$\pi_1 = \left\{\overline{a_1, a_7}; \overline{a_2, a_4}; \overline{a_3}; \overline{a_5}; \overline{a_6}; \overline{a_8}\right\}, \quad \pi_2 = \left\{\overline{a_1, a_3, a_4, a_6, a_8}; \overline{a_2, a_5, a_7}\right\},$$

$$\pi_3 = \left\{\overline{a_1}; \overline{a_2, a_6}; \overline{a_3}; \overline{a_4}; \overline{a_5}; \overline{a_7, a_8}\right\}, \quad \pi_4 = \left\{\overline{a_1}; \overline{a_2}; \overline{a_3}; \overline{a_4}; \overline{a_5}; \overline{a_6}; \overline{a_7}; \overline{a_8}\right\}.$$

Indeed, a partition π_i divides the points in the k-dimensional space into sets according to their i-th coordinate. For example, π_2 separates the black points from and white points in Figure 1.

Recall some basic definitions of the partition algebra. We say that π_i is *smaller* than π_j (denote $\pi_i \le \pi_j$) if each block of π_i is either equal to a block of π_j or is contained in a block of π_j. In our example, $\pi_4 \le \pi_1, \pi_4 \le \pi_2$ $\pi_4 \le \pi_3$, π_1, π_2 and π_3 are pairwise

incomparable. The relation *smaller* is a partial ordering on the set of partitions with a minimal element $\pi(0)$, where $\pi(0)$ is a null-partition including only blocks consisting of exactly one element of the set A. In our example, π_4 is equal to $\pi(0)$.

The product operation on the set of partitions is defined as follows:

If blocks of partition π comprise non-empty intersections of blocks of π_i and blocks of π_j, the partition π is called *a product* of π_i and π_j: $\pi = \pi_i \cdot \pi_j$

For example, $\pi_1 \cdot \pi_3 = \pi(0)$.

Consequence 2. Partition $P = \{Z_1, \ldots, Z_k\}$ is a detecting partition if and only if for every $i \in \{1, \ldots, k\}$, the following condition is satisfied:

$$\pi_1 \cdot \ldots \cdot \pi_{i-1}\pi_{i+1} \cdot \ldots \cdot \pi_k = \pi(0) \qquad (4)$$

Proof. Eq. (2) is equivalent to

$$\pi_1 \cdot \ldots \cdot \pi_{i-1}\pi_{i+1} \cdot \ldots \cdot \pi_k \le \pi_i \qquad (5)$$

It is obvious that (5) follows from (4), i.e., Eq. (4) is sufficient. The necessity of this condition follows from

$$\pi_1 \cdot \ldots \cdot \pi_i \cdot \ldots \cdot \pi_k = \pi(0), \qquad (6)$$

which is correct since all words of the set A are different. Let us replace π_i in Eq. (6) by the left portion of expression Eq. (5). If one of its π_i values is decreased, the product may decrease. This means that Eq. (4) holds, and therefore the consequence is proven.

It is easy to see that in Example 2 $\pi_1 \cdot \pi_2 = \pi_1 \cdot \pi_3 = \pi_2 \cdot \pi_3 = \pi(0)$, which means that the partition $P_1 = \{Z_1, Z_2, Z_3\}$ is a detecting partition and allows detecting any arbitrary error.

3.4. Partitioning for detection of unidirectional errors

Here we discuss the use of partitioning for detecting unidirectional errors. Let us call a difference $\delta(a_i, a_j)$ *monotonic* if a_i and a_j are comparable words $(a_i > a_j, \text{or } a_i < a_j)$. For example, $\delta(a_3, a_8)$ is monotonic, whereas $\delta(a_2, a_6)$ is non-monotonic. Theorem 2 states the necessary and the sufficient conditions that should be satisfied by partitioning capable of detecting unidirectional errors.

Theorem 2: Necessary and sufficient conditions for partitioning $P = \{Z_1, \ldots, Z_k\}$ to detect a unidirectional error in the words of the set $A = \{a_1, \ldots, a_M\}$ satisfying Eq. (3) for any two ordered words a_i and a_j of A (these words have a monotonic difference).

The proof of Theorem 2 is similar to the proof of Consequence 2 from Theorem 1.

It can be seen that Theorem 2 poses less strict requirements for partitioning, in comparison with the requirements of Consequence 2. For example, if the faults are

only unidirectional, then three non-zero elements (that are not denoted in bold in Table 3) should actually be replaced by zeros (see Table 4).

Table 4. Table of the monotonic differences

z_2	0					
z_3	0	0				
z_4	0	**1**	0			
z_5	0	**1**	0	0		
z_6	**1**	0	0	0	0	
z_7	**1**	**1**	0	0	0	0
	z_1	z_2	z_3	z_4	z_5	z_6

This measure allows reducing the number of partition blocks to two in the fault detecting partitioning. For example, unidirectional faults can be detected by partition $P_3 = \left\{\overline{z_1 z_2 z_3}, \overline{z_4 z_5 z_6 z_7}\right\}$. We must emphasize that the partitioning P_3 does not detect all faults according to the universal fault model. For example, an error in the circuit producing variables z_1, z_2, z_3 may lead to forming an erroneous codeword, $a_6 = 0111000$, instead of codeword $a_2 = 1011000$, and that fault will not be detected. Such a situation is impossible in circuits not comprising invertors since the error "011 instead of 101" is not unidirectional.

Note that partitioning of the functional circuit into separately implemented sub-circuits allows reducing the number of additional check bits. Various errors caused by a single fault can be detected by using just one additional check bit. The greater the number of additional check bits, the fewer the number of blocks required for constructing the detecting partition.

4. Algorithms of partitioning

In view of the discussion above, a task will optimally select the tradeoff between the number of check bits and the number of independent sub-circuits. Since these two criteria are contradictory and we do not have a general measure, which would allow their mutual weighting, it is worthwhile to consider two extreme cases. In each of them, one of the numbers to be minimized is considered as a limitation, which takes the minimal possible value. The first case is the two-block partitioning. In this case, the task is to find the solution with a minimal number of check bits. The second case is the multiple-block partitioning. In this case, at most, one check variable is used, and the number of partition blocks is to be minimized.

Note that if the set of codewords is complete, i.e., all 2^m binary vectors are used; any of the two extreme cases has a trivial solution. In the first case, the solution is to duplicate all variables and check by comparison. In the second case, it is to implement each variable by a separate independent circuit followed by the parity checking. If a specific set of words is incomplete, the trivial solutions can be further simplified.

In the present paper, both of the above-mentioned settings of the problems are discussed.

4.1. Algorithms for constructing two-block partitions

It is clear that if the number of codewords is M, they can be coded by $l = \log_2 M$ additional check bits. It is also obvious that a partition, one block of which comprises all the functional variables and the other block – all the check variables, satisfies Eq. (4) and thus constitutes the *detecting partition*. However, if not all the functional variables are independent, the number of check bits can be reduced. Assume that the set Y of functional variables can be divided into two blocks, Y_1 and Y_2, so that the variables from block Y_2 are functions of the variables from block Y_1. Now we have blocks Y_1 and Y_2 correspond to respective partitions π_1 and π_2 of the set of codewords according to the rules formulated above.

According to the proof of Consequence (2), $\pi_1 = \pi(0)$. If $\pi_2 \neq \pi(0)$, check bits are added to block Y_2 so that the partition π_2', corresponding to block Y_2 after adding check bits, is equal to $\pi(0)$ $\left(\pi_2' = \pi(0)\right)$. As a result, the encoded set of words will be divided into two blocks satisfying Eq. (4) and, consequently, this enables detection of any error in a word. In this case, the number of check bits is equal to $l' = \log_2 M'$, where M' is the maximal number of words in one block of partition π_2. Note that since $M' \leq M$, then $l' \leq l$.

Constructing the partitions with the minimal M' can be performed by simple enumeration. The enumeration can be shortened by determining a *core* – a set of variables in each block Y_1 that satisfies $\pi_1 = \pi(0)$. The core consists of variables that have different values in adjacent vectors of A^-.

In Example 2, the core is formed by variables y_1 and y_2. The core corresponds to partition $\pi_0 = \left\{\overline{a_1 a_4 a_8}; \overline{a_3 a_6}; \overline{a_2 a_5 a_7}\right\}$. Since the partition comprises a group of three elements, block Y_1 has to include at least two additional variables. For example, let us add y_3 and y_4 to variables y_1 and y_2. Accordingly, variables y_5 and y_6 belong to the second block Y_2. These variables correspond to partition $\left\{\overline{a_1 a_7}; \overline{a_8}; \overline{a_3 a_5}; \overline{a_2 a_4 a_6}\right\}$, $M' = 3$ and, consequently, at least two check bits are required. We denote them as z_7 and z_8: $P = \left\{\overline{y_1 y_2 y_3 y_4}; \overline{y_5 y_6 z_7 z_8}\right\}$. We may assume that $z_7 = y_1$ and $z_8 = y_2$. In this case, instead of six additional duplicated functional variables in the duplicated circuit of our example, we use only two functional variables.

4.2. Algorithms for constructing multiple-block partitions

We will now discuss methods for obtaining a multiple-block partition capable of detecting errors in binary words. To this end, we are going to use no more than one check variable minimize the number of blocks in the partition.

It should be clarified that if the set of binary words is complete (i.e., all 2^m binary vectors are in use); only a trivial solution is possible, namely, implementing each variable by a separate independent circuit and further parity checking. In this case, the number of blocks is equal to $k = m + 1$. If we deal with a specific incomplete set of

binary words, a smaller number of blocks in the partition may be obtained. Several such context-oriented solutions are presented below.

The following multiple-block partition algorithm is based on Consequence 1 of Theorem 1. First, if the set of words that is in use comprises adjacent vectors, they are encoded by one coding variable in such a manner that the number of non-zero positions in each code word is odd. Second, differences are determined between code words in all of the pairs. To accelerate the algorithm, absorbing differences are removed from the set of differences: if the differences δ_i and δ_j relate to one another as follows: $\delta_i \subseteq \delta_j$, then δ_j is removed from the set of differences. Let us assume that the differences form a set $\Delta = \{\delta_1, \ldots, \delta_H\}$ upon removing the absorbing ones. According to Consequence 1, the differences in the new set comprise information that limits partitioning the set of variables under check: no such differences can be part of one block of the partition.

The selection of the partition satisfying the limitations and comprising the minimal number of blocks can be performed by random enumeration. The algorithm comprises a) constructing a number of random partitioning versions satisfying the limitations, and b) selecting the best partition among them. The construction of each of the versions can be performed in n steps. Each of the steps is related to one of the variables under check and requires selecting the block of partitions for that specific variable. Enumeration of the variables is performed randomly, since the random selection of variables allows obtaining different solutions when repeatedly applying the same algorithm.

Assume that i variables forming a subset $Z^i \subset Z$ have been enumerated at the beginning of step $(i+1)$. And let the sub-set be divided into blocks $Z_1^i, \ldots, Z_k^i$. At step $(i+1)$, using the random equally probabilistic selection from the set of variables $Z \setminus Z^i$, which have not yet been selected, the next variable is selected. Let it be variable z^{i+1}. If there exists a block $Z_j^i, j = 1, \ldots, k$ such that for any $h = 1, \ldots, H$,

$$\delta_h \not\subset Z_j^i \bigcup \{z^{i+1}\},$$

then z^{i+1} will be attached to the block, $Z_j^{i+1} = Z_j^i \bigcup \{z^{i+1}\}$. If a number of such blocks exist, one of them (preferably the first found) should be selected. If the variable z^{i+1} cannot be attached to any of the earlier formed blocks, a new block is formed comprising only that specific variable: $Z_{k+1}^{i+1} = \{z^{i+1}\}$. Step $(i+1)$ is now terminated. Obviously, the partition obtained after n steps represents the detecting partition for words of the given set.

5. Experimental Results

One of the purposes of the present study was to estimate the efficiency of the above-described method of fault detection. The quality of the obtained solutions is estimated by a number of coding variables and the number of blocks in the partition. Since it is quite difficult to proportionally weigh and estimate the interrelation of these two parameters, the quality of the proposed solutions has been additionally estimated by the required overhead. For obtaining independent estimations, we used benchmarks and random functions. We also used the Mentor Graphics Leonardo Spectrum

synthesis tool and a CAD System "Synthesis 1" [18] developed for academic studies. Results of estimating some two-block partitioning versions are presented in section 6.1. Section 6.2 presents results of multi-block partitioning versions utilizing only one coding variable.

5.1. Evaluation results for two block partitioning

The experiments were conducted with "Synthesis 1". Combinational circuits forming output signals of sequential circuits were used as benchmarks. Such an approach can be explained by the fact that the set of codewords is strictly defined in the sequential circuits, and that in most cases this set essentially differs from the complete set $\left(M << 2^m\right)$. Estimation of a number l of check bits required for arbitrary error detection is shown in Table 5.

Table 5. Estimation of the number of check bits required for arbitrary error detection

Benchmark	M	m	$n1$	$n2$	l	overhead %
ACDL	19	27	13	15	1	3.7
ASS13	15	25	13	12	0	0
BIG	15	28	15	13	0	0
DORON	56	110	95	20	5	4.5
CAT	15	22	16	9	3	13.6
CPU	19	29	19	12	2	6.9
EX6	13	8	5	6	3	37.5
E2	16	18	14	8	4	22.2
E7	17	20	15	8	3	15
E17	14	17	11	9	3	17.6
KOBZ	54	53	40	17	4	7.5
LIOR	27	31	27	9	5	16.1
PP	15	28	14	14	0	0
SASI	57	54	44	14	4	7.4
SOL	60	68	59	14	5	7.4
v16	12	18	12	8	2	11.1
v110	13	18	11	9	2	11.1
v1120	17	29	14	15	0	0
Average	25	34	24	12	2.6	10

Table 6. Overhead estimation

Benchmark	t	number of gates			
		g_0	g_1	g_2	overhead %
ACDL	21	434	415	418	91.9
ASS13	10	102	85	92	73.5
BIG	23	264	252	249	89.8
DORON	28	321	295	318	91
CAT	15	70	67	63	85.7
CPU	18	90	84	85	87.8
EX6	9	69	68	65	92.7
E2	48	200	166	186	76
E7	17	151	143	135	84.1
E17	12	63	54	58	77.8
KOBZ	24	238	226	221	87.8
LIOR	29	225	207	234	96
PP	25	188	169	164	77.1
SASI	25	233	219	218	87.5
SOL	30	449	385	446	85.1
v16	19	183	133	175	68.3
v110	20	263	183	253	65.8
v1120	19	311	280	298	85.8
Average	22	214.1	190.6	204	84

Columns in Table 5 are as follows:

M – the number of codewords, m – the number of variables being checked, n_1, n_2 – the number of variables in the first and in the second blocks of partition, respectively, l – the number of coding variables $n_1 + n_2 = m + l = n.$

Overhead is calculated in percents and reflects the relative increase in the number of variables**.** The table shows that the number of check bits is smaller by about 10% than the number of variables under check. Some schemes exist, where detecting any errors in a word is possible without using additional (check) variables. The experimental results therefore confirmed that the proposed algorithms are efficient from the point of reducing the number of check bits.

Table 6 presents estimations of additional overhead caused by the partitioning of the scheme into separate independent sub-schemes and by increasing the number of outputs.

In Table 6, t is the number of inputs and g_i is the numbers of gates in the scheme. Index 0 indicates characteristics of the initial scheme, whereas indexes 1 and 2 denote two separate sub-schemes into which the initial scheme is partitioned. Overhead reflects the relative increase in hardware in percents.

In comparison with the duplication, overhead is decreased by about 15%. In some schemes, the overhead reduction reaches 30% and even more.

5.2. Evaluation results for multiple block partitioning

This study was performed using statistical methods. We have developed a generator of pseudo-random codewords enabling random equally-probable selection from a set W_m of the subset $A \subset W_m$, comprising M words. The selection was performed so that none of the m variables under check takes a constant (zero or non-zero) value in all codewords. For each of the generated sets of codewords, a partition with a minimal number of blocks was selected. In each point (m, M), $m \in \{4,6,8,10\}$, $M \in \{4,6,8,10,20,40\}$, and an average number k^* of blocks was calculated (by averaging over 100 experiments). Results of the experimental study are presented in Table 7.

Table 7. Statistical relation between the number of blocks in detecting partitions and the number of check variables and the number of codewords.

M \ m	4	6	8	10
4	2,40	2,09	2,16	2,92
6	3,19	2,45	2,39	2,80
8	4,37	3,43	2,72	2,95
10	5,00	3,63	2,92	3,00
20		6,82	4,20	3,31
40		7,00	8,34	4,79

Figure 1 illustrates the experimental results in a more visual manner. The graph reflects how the average number m/k^* of variables under check in a block depends on the relative number $q = M/2^m$ of code words. As was expected, the number of blocks in a partition grows as q increases, reaching the maximal value of $k = m+1$ when $q \cong 0.5$. When $q < 0.1$, the average number of elements in the block is no less than two, i.e., $k \le 0.5m$.

In other words, the proposed method of detecting faults is advantageous exactly in that area. Note that the mentioned area is the area of practical interest. For example, in MCNC benchmarks in all cases when $m \ge 7, q \le 0.1$.

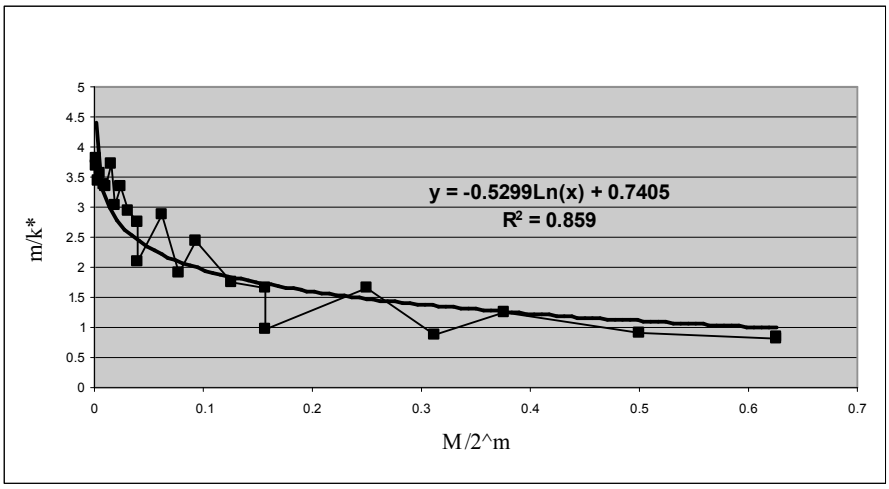

Fig. 1. Relation between the average number m/k^* of functional variables in a single block and the relative number $q = M/2^m$ of codewords.

In order to address the question of increasing complexity of a circuit upon dividing the circuit into independent sub-circuits, a number of experiments have been performed using the CAD Leonardo Spectrum. In the experiments, pseudo-random functions of 6 or 8 variables were generated; parameters M and m were varied within the above-mentioned ranges. Each of the functions was implemented in the form of FPGA in two versions: with and without dividing them into sub-circuits. Both of the versions were compared by their area overhead. The results show that the complexity of the circuit increases with the growth of q. More specifically, the factor of complexity varies from 1.3 for $q = 0.02$ up to 1.7 for $q = 0.16$, and the numbers indicate that the proposed method seems to be less complex than the duplication.

6. Conclusions

In the present paper we have introduced a new method for designing concurrent checking logic circuits. This method is characterized by two features.

First, the method combines the redundant encoding with the partitioning of the circuit into separate independently implemented sub-circuits. The partitioning may be adjusted either by increasing the number of coding variables or by reducing the number of blocks in the partition.

Second, the method is context-oriented, i.e., selection of the code and the partition is performed for a given specific set of codewords to be checked.

We formulated theoretical fundamentals of the proposed design method. Based on the fundamentals, we have solved the problem of optimal partitioning on the set of functional outputs. The main criteria of the optimal partitioning are as follows: a) a minimized number of check bits, and b) a minimized number of partition blocks. We introduced a method for handling a trade-off between these criteria for obtaining the optimal solution.

The method has proven to be efficient for detecting both arbitrary and unidirectional errors.

Benchmark experiments, together with the Monte Carlo statistical experiments, show that the average number of check bits required for detecting arbitrary faults can

be estimated as close to 10% of the number of functional variables. In the majority of the performed experiments with multi-block partitioning, the required hardware overhead did not exceed 50%.

The newly formulated theoretical fundamentals and the design method utilizing these fundamentals form a new approach for designing on-line checking circuits. This approach allows obtaining solutions that can be flexibly optimized according to various criteria.

7. References

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

[2] 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.*

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

[4] Ostrovsky V. and Levin I., 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.*

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

[6] Smith J. E., On Separable Unordered Codes*, IEEE Transaction on Computers, Vol. Vol. 33, No. 8, pp. 741-743, Aug. 1984.*

[7] Ostrovsky V., Levin I., Ostanin S., Synthesis of Self-checking Controllers Based on Modified (m, n)-cod, *Automatic Control and Computer Science, Vol. 37, No. 2, pp. 48-55, 2003*

[8] Jha N.K. and Wang S.-J., Design and Synthesis of Self-Checking VLSI Circuits, *IEEE Transaction CAD, Vol. 12, No. 6, pp. 878–887, Jun. 1993*.

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

[10] Pomeranz I. and Reddy S.M., 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.*

[11] Bose B. and D. J. Lin. Systematic Unidirectional Error-Detecting Codes, *IEEE Transaction on Computers, Vol. 34, pp. 1026-1032, Nov. 1985.*

[12] Omana M., Losco O., Metra C., Pagni A., On the Selection of Unidirectional Error Detecting Codes for Self-Checking Circuits Area Overhead and Performance Optimization*, Proceedings of the 11th IEEE International On-line Testing Symposium, pp. 163-168, Jul. 2005.*

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

[14] Saposhnikov V.V., Morosov A., Saposhnikov Vl.V., Gössel M., 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.*

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

[16] Levin I., Ostrovsky V., Keren O., Sinelnikov V., Cascade Scheme for Concurrent Errors Detection*, Proc. of 9th EUROMICRO Conference on Digital System Design (DSD'06) pp. 359-368, 2006*.

[17] Hartmanis J. and Stearns R. E., Pair Algebra and Its Application to Automata Theory, *Information and Control, Vol. 7, pp. 485-507, 1964*.

[18] Baranov S., CAD System for ASM and FSM Synthesis, *Lecture Notes in Computer Science, Vol. 1482, "Field-Programming Logic and Applications", Springer-Verlag, Berlin Heidelberg, pp. 119-128, 1998.*