Implementation of Concurrent Checking Circuits by Independent Sub-circuits[1]

Vladimir Ostrovsky and Ilya Levin
School of Engineering, Bar-Ilan University, Ramat Gan 52900, Israel

Abstract

The present paper proposes a new method for detecting arbitrary faults in a functional circuit when the set of codewords is limited and known in advance.

The method is based on implementation of the functional circuit by a plurality of separate independent sub-circuits. Each of such sub-circuits generates its own subset of output signals. Since the sub-circuits do not have common elements, any single fault may result in errors only in one of the subsets.

The paper presents a solution of the problem of optimal partition of the set of output variables into independent subsets. A number of properties of partitions are proven. The proposed algorithms of the optimal partition are based on these properties.

A scheme of the checker for the proposed self-checking approach is presented. Benchmarks' results indicate efficiency of the described technique.

1. Introduction

Numerous publications concerning on-line checking of output words of discrete circuits present serious results but the problem still remains non-solved. The problem of developing highly reliable and hardware economical methods for detecting faults (errors) remains the actual and important practical task.

For the sake of simplicity, the task of output checking we consider only for combinational circuits. Faults (temporary or permanent) may occur in the functional circuit and may lead to distortion of one or more positions of an output codeword. Traditionally, it is supposed that the time interval between any two faults is sufficient for eliminating consequences of the first fault before the second fault occurs. Therefore, only one fault may be present in the circuit at given time. This assumption is used for constructing models of acceptable distortions of output words.

Many publications use the following two models of possible errors distortions or their combinations. The first model is that the system of logic functions, describing a functional circuit, is monotonic. For example, circuits without invertors suit to that assumption. In such circuits, any single fault may lead only to unidirectional errors in an output codeword. In [1] a synthesis method for inverter-free circuits is described. Another solution that is based on duplication of some elements of the circuit is proposed in [2] where a method for the design of unidirectional combinational circuits is proposed.

The second model supposes that the number of distorted bits in the word cannot exceed a predetermined threshold "t" [3]. Such a model can be possibly applied to circuits where the splitting coefficient of the input gates is relatively small, and each of the gates participates in forming of a relatively small number of output signals. There are circuits where using the above models is non-sufficient and the duplication based solution [4-6] is preferable.

Majority of the known concurrent checking schemes suppose 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 necessary and rational to construct a so-called "context-oriented" concurrent checking scheme where the set of possible codewords is limited and known in advance.

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

In present paper we use the context-orientation of specific circuits in a new concurrent checking scheme that detects any arbitrary errors.

For this aim, we present a concurrent checking scheme based on dividing the functional circuit into a number of separate independent sub-circuits. Each of such sub-circuits generates its own subset of output signals. Since the sub-circuits do not have common elements, any single fault may result in errors only in the subset of the output signals.

The paper is organized as follows. Section 2 presents some important definitions and a brief overview of related error-detecting techniques. In Section 3 the detecting partition concept is introduced and a number of its properties are described and proven. Algorithms of optimized partition on the set of output variables into a two subsets are presented in Section 4. Design of the checker for the proposed scheme and estimation of its complexity is presented in Section 5. Experimental results are presented in Section 6. Conclusions are provided in Section 7.

2. Definitions and Related Works

Let $Y = \{y_1, \ldots, y_m\}$ be a set of output variables of the functional circuit to be checked. We will call these variables as functional variables. Let, during normal functioning, the variables take values from a set of codewords $A = \{a_1, \ldots, a_M\}$. We say that the set of codewords is *complete* if $M = 2^m$ and *incomplete* if $M < 2^m$. The complete set of ***m***-bit codewords we denote W_m.

In the general case, we define a fault model by operator ψ on the following way. Let, a codeword $a_i \in A$ takes an erroneous value from the set $\psi(a_i)$ as a result of a fault in the circuit. Let $\psi(\mathrm{A}) = \bigcup_{\mathrm{a_i} \in \mathrm{A}} \psi(\mathrm{a_i})$.

According to the fault-secure property, the set of words A allows detecting erroneous values designated by the operator ψ, if and only if:

$$\psi(\mathrm{A}) \bigcap \mathrm{A} = \varnothing \qquad (1)$$

Let us define a function $R = F(Y)$ on the set of functional variables, wherein the function R allows distinguishing a codeword from a non-codeword. Function R must take two alternative values: one value on the set of codewords A, and the other value on the set of non-codewords $\psi(A)$. We will call this function a *checking* function. To be more specific, let us assume $F(a) = 1$ if $a \in A$, and $F(a) = 0$ if $\mathrm{a} \in \psi(\mathrm{A})$. The function allows arbitrary values on the vectors not belonging to the two mentioned sets.

The checking function is implemented by a specific circuit called *checker*. The checker must have the self-testing property i.e., for any single fault of the checker at least one codeword $a \in A$ must exist, for which $R = F(a) = 0$. In order to ensure this property, the value of the checking function, as usual, are presented by two signals R_1 and R_2, $\mathrm{R} = \mathrm{R_1} \oplus \mathrm{R_2}$.

If A is incomplete, construction of the self-testing checker presents a complex and actually, almost non-resolvable task. It is known that all universals schemes of checkers may become non-self-testing if the set of code words is incomplete. Actually, these schemes become non universal and must be adapted to the given set of code words. Sometimes, the set of code words is such that one cannot check reaction of the checker to non-code words by using only the codewords. Owing to that fact, a number of works (for example [7]) suggests neglecting the above requirement and testing the checker in a specific mode by using both the codewords, and non-codewords.

Here we use a known scheme of a checker [8], where the authors propose to implement the checker for the context-oriented concurrent checking schemes in a form of "sum of minterms" (SOM) of output codewords. It is important to emphasize that, according to [8], the SOM-checker examines whether an output vector belongs to the set of possible codewords, contrary to a traditional checker that checks some particular features that differ any codeword from any non-code word (for example, whether the number of "high" bits suits to the combination of check variables in the Berger code [9]).

If the fault model on the set $\psi(A)$ specifies some limitations then the condition (1) can be fulfilled by using suitable error-detecting coding methods. In this paper, we study the case $\psi(A) = W_m$, where no limitations on the set $\psi(A)$ are specified i.e., any arbitrary error may occur. In such a case there is no such a code which would ensure fulfillment of the condition (1), and therefore there is no code that allows distinguishing a codeword from a non-codeword. In such cases, the checking is usually based on duplication [4-6], but the two functional circuits of the scheme must have no common elements and the fault-secure property is preserved as long as both of the circuits do not produce identical errors.

Duplication based solutions are neither the only possible, nor the most efficient for schemes where the redundant coding is combined with independent implementation of functional variables. For example, more economic solutions are obtained based on the parity check and are described in [10-12]. The main idea investigated in these papers is to divide the set of check variables into groups. Each group comprises variables implementable by independent schemes (i.e., those having no common elements). Owing to that, any single fault may result in only one error in each group. Such error can be detected by the parity check, by using one coding variable which fulfils each group. Paper [11] proposes a method for optimal partitioning the variables into groups, and a method of transforming the schemes under check.

In our paper, contrary to the above-mentioned works, we use independency between groups of variables, and not between separate variables. In such schemes, any fault may lead to errors only among variables of one and the same group. Such an approach was investigated in [12]. However, upon conducting benchmarks tests, the Authors have come to the conclusion that the optimal partition for this case is a partition where each variable under check forms a separate group, i.e., is implemented by a separate independent scheme. Contrary to [10-12], the present work describes a method that does not use the parity check and reduces a number of check bits for the same number of groups. Alternatively, the proposed method allows decreasing the number of groups for the same number of check bits.

There is one additional important difference of the proposed method over the methods developed in [10-12]. These methods use modification of circuits preliminary synthesized by using commercial CAD systems. Such modified schemes should be further verified and examined, which has to be considered as a disadvantage of the above methods. In view of this, methods which use only the standard CADs and do not require modification of the schemes, are preferable. One of such methods is in the focus of the present paper.

3. Error Detecting Partitions and their Properties

Let, in a general case, $(n-m)$ check variables are added to the functional variables $y_1,\ldots,y_m$. Let the common set of the variables (including those under checking and the additional ones) is denoted $Z = \{z_1,\ldots,z_n\}$. Let the set Z is partitioned into k blocks $Z_1,\ldots,Z_k$ so, that the variables of every block are implemented by independent circuits. In this partition, two circuits are considered independent if they do not comprise common logical elements. However, it should be emphasized that 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 compatible the elements $b_i \in B_i$, and $b_j \in B_j$ $i,j \in \{1,\ldots,k\}$ if corresponding values are used in at least one of the codewords. In a specific case, the compatibility relation can be a function. If $b_i \in B_i$ is a function of $b_j \in B_j$, it means that values of the variables of the set Z_i, in all the codewords, are uniquely defined by the variables belonging to the set Z_j. If such a relation exists, we will write $Z_i = F(Z_j)$.

Theorem 1. Partition $P = \{Z_1, \dots, Z_k\}$ represents the *detecting partition*, if and only if for each block $Z_i \in Z$ the following condition (2) is correct:

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

Proof: Let the condition (2) is fulfilled, and let (due to a fault in the circuit) an erroneous vector $\hat{b}_i$ is formed instead of the correct vector $b_i \in B_i$. Then, either $\hat{b}_i \notin B_i$, or (as a result of (2)) $A(b_i) \cap A(\hat{b}_i) = ø$ where $A(b_i)$ and $A(\hat{b}_i)$ are sets of codewords, the *i*-th field of which has values of b_i and $\hat{b}_i$ respectively. In both cases the condition (1) is satisfied and, consequently, any erroneous word differs from a codeword.

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

Let us formulate some consequences of the theorem, which are useful when constructing and analyzing partitioning detecting errors.

Let us define a *difference* $\delta(a_i, a_j)$ between two words a_i and a_j to be a set of variables that have different values in these two words.

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

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

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

The proof is not presented here in view of its simplicity.

It should be noted that if the difference between two codewords comprises exactly two variables, in any detecting partitioning these two variables must belong to different blocks (according to condition (3)). If A comprises at least one pair of words differing one from another by the value of only one binary variable, 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 the words no less than two. This conclusion is well supported by the known results of the coding theory. However, in a general case, partitioning of a circuit into separately implemented sub-circuits opens new ways for detecting errors and thus changes the coding requirements. For example, for detecting errors in not more then t bits the requirement of the minimal distance $t+1$ between the words becomes non obligatory.

Notice that each block Z_i $(1 \le i \le k)$ of the partition $P = \{Z_1, \dots, Z_k\}$ defines a partition π_i of the set A of codewords. A certain block of π_i comprises all the codewords having the same values of variables from the set Z_i .analogy with codes, let us call a partition as *detecting partition* i.e., partitions detecting the errors defined by the function ψ, if and only if the set $\psi(A)$ of erroneous words meets the condition (1) while implementing the variables of each block by a separate independent circuit.

Let B_i be a set of values which are taken by the variables from the block Z_i, $i \in \{1, \dots, k\}$ in the codewords. Variables of the block Z_i form *i-th field* of the codeword. The codewords define a compatibility relation between elements of any two sets B_i and B_j. We will call

Consequence 2. Partition $P = \{Z_1, \ldots, Z_k\}$ is the detecting partition if and only if for every $i \in \{1, \ldots, k\}$ the following condition (4) is satisfied:

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

where $\pi(0)$ is a null-partition, including only blocks consisting of exactly one element.

Proof. The condition (2) is equivalent to

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

It is obvious that (5) follows from (4) i.e., condition (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 (6) with the product from the left portion of the expression (5). If one of its members is decreased, the product may decrease. This means (4), and therefore the consequence is proven.

Summarizing the present section, it should be noted that partitioning of the functional circuit into separately implemented sub-circuits allows reducing the number of additional check bits. For example, various errors being consequences of a single fault can be detected, by using just one additional check bit. It is clear that the greater the number of additional check bits, the lesser the number of blocks required for constructing the detecting partitioning.

In view of the discussed above, a task arises to optimally select 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 common measure which would allow their mutual weighting, it is worthwhile considering two extreme cases. In each of them one of the numbers to be minimized is understood as a limitation, which takes the minimal possible value. In the first case (the minimum of check bits) the number of check bits does not exceed 1 and the number of partition blocks is to be minimized. In the second case (the minimum of separate independent sub-circuits) the number of blocks is equal to two, and looks for the solution where the number of check bits is minimal. It should be noted that if the set of codewords is complete i.e., all the 2^m binary vectors are used; any of the two extreme cases has a trivial solution. In the first case it is implementation of each variable by a separate independent circuit and checking by *modulo 2*. In the second case, the solution is duplication of all variables and checking by comparison. If a specific set of words is incomplete, the trivial solutions can be further simplified.

In the present paper we concentrate only on a two-block solution with a minimal number of check bits. Below we deal with the algorithm for the solution of this problem.

4. Algorithms for Partitioning a Circuit into two Independent Sub-circuits

We solve a task of coding and simultaneously a task of creating a partition $P = \{Z_1, Z_2\}$, that satisfies a condition (4), i.e., such a partition for which the following statements are correct: $\pi_1 = \pi(0)$ and $\pi_2 = \pi(0)$. We obviously prefer solutions in which the number of coding variables is minimal. Please keep in mind that: $Z = Z_1 \cup Z_2 = \{z_1, \ldots, z_n\}$- is a set combining the controllable $(z_i = y_i, i = 1, \ldots, m)$ and the coding $(z_{m+1}, \ldots, z_n)$ variables; π_1 and π_2 - are partitions of set A of code words which can be defined by blocks Z_1 и Z_2, respectively. The following algorithm can be applied for solving the above task.

1. By enumeration, build minimal sub-sets of functional variables $Y' \subseteq Y$, which satisfy (4). Let us consider the sub-set to be minimal if none of the variables can be removed there-from without infringing the condition (4). The enumeration can be minimized by defining a so-called "kernel" i.e., a sub-set of such variables which obligatory belong to any Y', which satisfies the above-mentioned conditions. The following criterion can be used for revealing

the "kernel variables": The variable belongs to the kernel if and only if two code words exist which differ one from another only by the value of that variable.

2. Let $Z_1 = Y'$ and $Z_2 = Y \setminus Y'$. If $\pi_2 = \pi(0)$, then the partition is built and no additional coding is required. In the opposite case, we define a number l of coding variables: $l = \log_2 M'$, wherein M' - is the maximal number of variables in one block π_2.

3. By using enumeration, we define Y', which corresponds to the minimal l. We include l additional coding variables in the block Z_2, and assign such values to these l variables, that $\pi_2 = \pi(0)$. Uncertaincy that occurs in this step could be used for simplifying the checker or the scheme to be checked.

We demonstrate below the process of constructing two-block partitions on the set of words from the Example 1.

Example 1. The system of the five functions of three variables $x_1, x_2\ x_3$ presented in Table 1.

Table 1

	x_1	x_2	x_3	y_1	y_2	y_3	y_4	y_5
a_1	0	0	0	1	0	1	1	0
a_2	0	0	1	0	0	0	0	0
a_3	0	1	0	0	1	0	1	0
a_2	0	1	1	0	0	0	0	0
a_4	1	0	0	1	1	0	1	0
a_2	1	0	1	0	0	0	0	0
a_2	1	1	0	0	0	0	0	0
a_5	1	1	1	0	0	0	1	1

Table 2

	y_1	y_2	y_3	y_4	y_5		
	Z1	Z2	Z3	Z4	Z5	Z6	Z7
a_1	1	0	1	1	0	1	1
a_2	0	0	0	0	0	0	0
a_3	0	1	0	1	0	0	0
a_4	1	1	0	1	0	1	1
a_5	0	0	0	1	1	0	1

The core is formed by a variable y_1 (from the adjacent sets $a_4 = 11010$ and $a_3 = 01010$). The core corresponds to the partition $\pi_0 = \{\overline{a_1 a_4}, \overline{a_2 a_3 a_5}\}$. Since the partition comprises a group of three elements, the block Y_1 must comprise the variables of the core and at least two additional variables, for example, y_2 and y_5. In view of that, variables y_3 and y_4 belong to the second block Y_2. These variables correspond to the partition $\{\overline{a_1}, \overline{a_2}, \overline{a_3 a_4 a_5},\}$, $M' = 3$ and, consequently, at least two check bits are required. We denote them z_6 and z_7. $P = \{\overline{y_1 y_2 y_5}, \overline{y_3 y_4 z_6 z_7}\}$. One version of coding is presented in Table 2.

In this example, only two redundant check bits are necessary for detecting arbitrary errors - instead of five variables that would be required in the duplication based solution.

The initial circuit and the circuit corresponding to the two-block partition are presented in Figure 1 (a) and (b).

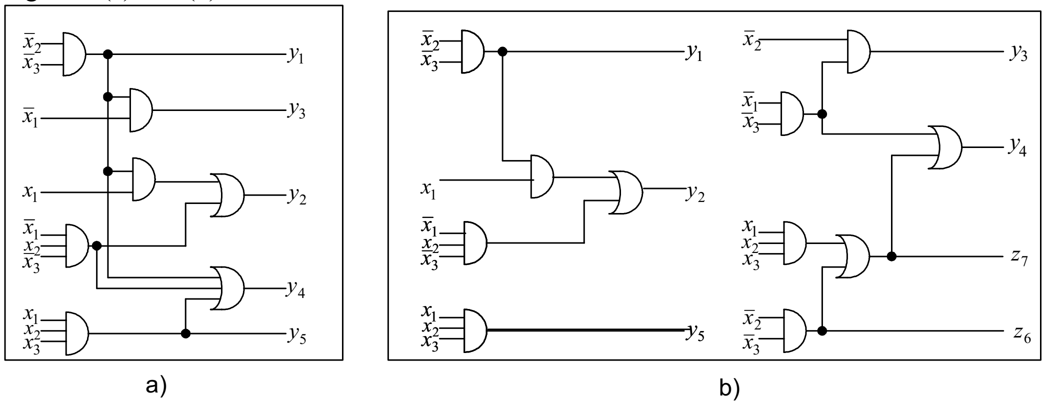

Figure 1. Initial circuit (a) and its Implementation by two independent portions (b).

It is clear from Figure 1, that complexity of the given circuit is "equal" to 7 gates (17 inputs). The complexity of the circuit increases to 11 gates (25 inputs) after partitioning it into two independent circuits. As a result, in the above example, the proposed scheme detecting any fault requires little over 50% overhead.

5. Synthesis of a checker and estimation of its complexity

The problem of designing self-checking checkers, in the case when the set of possible codewords is limited, is known as difficult and sometimes even insolvable; particularly, the comparator usually utilized in the duplication solution, does not have the self-checking property. We assume that the checker is tested in a specific mode during the period of functioning or maintenance.

We propose to design the checker in a form of (n,2)-circuit. Set of inputs of the circuit are variables to be checked: $\{z_1,\dots,z_n\}=\{y_1,\dots y_m, z_{m+1},\dots, z_n\}$. Outputs of the circuit R_1 and R_2 take alternative values (0,1) on the set A of the codewords, and equal values on the set of non-codewords. Let us partition the set of codewords into the two subsets $A=\{A_1, A_2\}$ and assume that $R_i=1$ only on the vectors from the set $A_i.i=1,2$. The criterion of partitioning will be simplification of the scheme of checker (reduction of the overhead). Since the products corresponding to codewords do not adjacent pairwise, the simplification can be reached only by using a multi-level synthesis. Fig. 2 illustrates a multi-level scheme of the checker, for the case of two-block partition (Example 1, Fig. 1). $A_1=\{a_1a_4\}, A_2=\{a_2a_3a_5\}$.

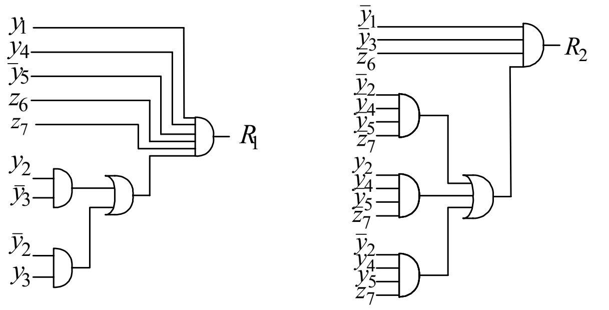

Figure 2. The SOM-checker for the two-block partition

As can be seen in Fig. 2, the scheme of checker comprises nine gates with the total number of inputs equal to 31. It should be noted that, according to the duplication solution, the checker would be more complex- it would comprise 24 gates, each having two inputs.

In a general case, for estimating complexity of a SOM –checker, one may use the following known equation for the number of inputs S_1: S_1=n(M+1). The general case means that the simplification due to the factorization is neglected. A similar estimation for a duplication based scheme of the checker can be estimated as: S_2=12(m-1). Comparison of the expressions S_1 и S_2 shows that the SOM-checker, is simpler from the point of inputs for schemes where the number of codewords is relatively small i.e., smaller than 12. By checker's minimization, this threshold can be slightly shifted towards a greater number of codewords.

From the point of the number of gates, the SOM checker is always a winner. At the same time, the number of inputs at specific gates may be enormously great. However, if the number of inputs is limited by any really implementable value (say, eight) and both *n* and $M \le 50$, the SOM checker has less or comparable number of gates then the duplication scheme checker.

The above-presented estimations may suggest some limitations to applicability of the proposed method of checking of outputs. One should take into account, however, that the present paper does not set/solve the task of the checker simplification. At the stage of coding, quality of solutions is traditionally estimated by the number of check bits. Therefore, would selection of codes in our algorithms (which is now rather arbitrary) be used for simplifying the checker, the field of applicability of the algorithms could be broadened.

Estimations of the number of check bits, and of the complexity of the schemes under control will be presented in the next section.

6. Experimental Results

The research was conducted with CAD System "Synthesis" [13]. 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 $\left(M << 2^m\right)$. Estimates of the number l of check bits that are required for detecting a arbitrary error in an output word are presented in Table 3.

Table 3

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	3	10

Table 4

Benchmark	t	number of gates				number of inputs			
		g_0	g_1	g_2	overhead %	s_0	s_1	s_2	overhead %
ACDL	21	434	415	418	91.9	1225	1174	1176	92
ASS13	10	102	85	92	73.5	262	216	236	73
BIG	23	264	252	249	89.8	707	685	671	92
DORON	28	321	295	318	91	821	752	830	93
CAT	15	70	67	63	85.7	173	167	157	87
CPU	18	90	84	85	87.8	218	205	207	85
EX6	9	69	68	65	92.7	173	172	164	94
E2	48	200	166	186	76	558	467	522	77
E7	17	151	143	135	84.1	402	379	367	86
E17	12	63	54	58	77.8	157	133	143	76
KOBZ	24	238	226	221	87.8	588	566	575	94
LIOR	29	225	207	234	96	559	514	608	101
PP	25	188	169	164	77.1	518	475	452	79
SASI	25	233	219	218	87.5	613	565	599	90
SOL	30	449	385	446	85.1	1117	956	1133	87
v16	19	183	133	175	68.3	489	358	469	69
v110	20	263	183	253	65.8	677	489	658	69
v1120	19	311	280	298	85.8	793	722	764	87
Average	22	214	191	204	84	558.3	500	541	85

Columns in Table 3 are:
M – the number of codewords, m – the number of variable being checked, n_1, n_2 – respective numbers of variables in the first and in the second blocks of partition, $n_1 + n_2 = m + l$.

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

Table 4 presents estimations of additional overhead caused by partitioning of the scheme being controlled into separate independent sub-schemes and by increasing the number of inputs.

In Table 4, t is the number of inputs, g_i and s_i - are the numbers of gates in the scheme and the numbers of their total inputs, respectively. Index 0 indicates characteristics of the initial scheme, while indexes 1 and 2 marks two separate sub-schemes into which the initial scheme is partitioned. Overhead reflects the relative increase of 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.

7. Conclusions

We have presented a new approach for designing a concurrent checking circuit for detecting arbitrary errors. The method is based on assumption that the functional circuit to be checked may produce a limited set of possible output codewords. This fact gives a specific opportunity to implement the circuit in a form of independently functioning sub-circuits. Each of these sub-circuits implements a specific subset of functional outputs. The independent implementation of the sub-circuits guarantees that a single fault, if occurs, affects outputs of only one of the sub-circuits.

The paper formulates and solves the problem of optimal partitioning on the set of functional outputs. The main criterion for optimal partitioning is a number of check bits, which should be kept minimal. Two-block partitions were investigated in more details.

Benchmark experiments show that the average number of check bits required for detecting arbitrary faults can be estimated as about 10% of the number of functional variables. The average overhead estimated for these cases is of about 85%.

Regardless specific values which characterize results of applying the developed algorithm to a set of benchmarks, a conclusion can be made that one should consider error-detectable coding in the close connection with partitioning the scheme to be checked into independently implementable subcircuit. Such a partition allows reducing both the number of coding variables, and the hardware overhead.

The method can be used for checking of sequential schemes, and may also simplify the schemes capable of detecting unidirectional faults.

Acknowledgments

The Authors are very grateful to Prof. Samary Baranov and to Mr. Yuri Tsukerman for their help in conducting the benchmark tests.

8. References

[1] 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, 1993.*

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

[3] Bose, B. and D. J. Lin, Systematic Unidirectional Error-Detecting Codes, *IEEE Trans. Comp.*, pp. 1026-1032, Nov. 1985.

[4] Sedmak, R. M. and H. L. Liebergot, Fault-Tolerance of a General-Purpose Computer Implemented by Very Large Scale Integration, *Proc. FTCS*, pp. 137-143, 1978.

[5] Kraft, G. D. and W. N. Toy, *Microprogrammed Control and Reliable Design of Small Computers, 1981.*

[6] Sellers, F., M-Y Hsiao and L. W. Bearnson, *Error Detection Logic for Digital Computers*, McGraw-Hill Book Company, 1968.

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

[8] Levin I., Karpovsky M. (1998). On-line Self-Checking of Microprogram Control Units. *4-th IEEE International On-line Testing Workshop, Capri, Compendium of papers, 153-159.*

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

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

[11] V. V. Saposhnikov, A. Morosov, VL. V. Saposhnikov, M. Gössel. Design of Self-Checking Unidirectional Combinational Circuits with Low Area Overhead. *Proc. 2nd Int. On-Line Testing Workshop, 1996.*

[12] 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, June 1994.*

[13] S. Baranov: CAD System for ASM and FSM Synthesis. *FPL 1998,* pp. 119-128.