DOI: http://doi.org/10.25728/cs.2024.4.3

DESIGN OF SELF-CHECKING DIGITAL DEVICES WITH BOOLEAN SIGNALS CORRECTION USING WEIGHT-BASED BOSE–LIN CODES

D. V. Efanov*,** and Y. I. Yelina*

*Peter the Great Saint Petersburg Polytechnic University, St. Petersburg, Russia
**Russian University of Transport, Moscow, Russia

*✉ TrES-4b@yandex.ru, **✉ eseniya-elina@mail.ru

Abstract. This paper proposes a method for designing self-checking digital devices with Boolean signals correction and weight-based Bose–Lin codes. Unlike previous studies, the method involves Boolean signals correction (BSC) in the concurrent error-detection (CED) circuit for those functions describing the outputs of source devices that participate in the formation of data symbols of weight-based Bose–Lin codes. In such codes, as in the absolute majority of uniform separable codes, several data vectors correspond to the same check vector; therefore, it is possible to choose a method for determining BSC functions. We describe an algorithm for determining their values for each input combination, considering the testability of the checker and transformation elements in the CED circuit. The method involves the so-called "base" structure for monitoring multi-output devices by output groups. With this method, the designer of a self-checking device has high variability in choosing the design method and can regulate important indicators (structure redundancy, controllability, energy consumption, and others). Experiments with combinational benchmarks from MCNC Benchmarks were carried out. According to the experimental data, the method has high efficiency in terms of structure redundancy compared to the duplication method widespread in practice. The method can be effective when designing real devices with fault detection used in all areas of technology, including critical application systems in industry and transport.

Keywords: self-checking device, concurrent error-detection circuit, Boolean signals correction, weight-based sum code, weight-based Bose–Lin code.

INTRODUCTION

When designing highly reliable and safe discrete blocks and nodes of critical application systems, it is important to ensure the timely detection of faults and computational errors arising during their operation [1–4]. This property is achieved by off-line and on-line testing equipment [5–7].

One approach to designing discrete systems with fault detection is the development of devices with testable and self-checking structures [8–12]. This requires the introduction of redundancy, according to certain principles, into a source device (further called an object under diagnosis) or the use of external technical diagnosis means, including the equipment of objects under diagnosis with self-checking concurrent error-detection (CED) circuits [1, 13].

CED circuits are designed so that it is possible to judge the correct operation of an object under diagnosis by the values of the functions computed by this object or the functions formed in its structure at special checkpoints. Physical defects lead to the occurrence of faults at the outputs of the internal structure elements of an object under diagnosis and, as a consequence, to the appearance of distortions in the values of computed functions, which is detected by a CED circuit. For example, diagnostic attributes are the belonging of binary vectors formed in a CED circuit to a set of codewords of any predetermined redundant binary codes [1] or the belonging of functions formed in

a CED circuit to a predetermined special class of Boolean functions (e.g., linear, monotonic, or self-dual) [14, 15]. Several diagnostic features can be used together and simultaneously [16, 17].

There exist two main approaches to organizing CED circuits for discrete devices. The first (classical) one consists in that an object under diagnosis is equipped with a CED circuit in which the data vector formed at the object's outputs is supplemented with the check vector computed using a check logic block [1]. A checker is installed to verify the mutual correspondence of the data and check vectors. The second approach implies correction of the functions computed by the object under diagnosis in the CED circuit so that the codeword of a preselected redundant code is formed or the functions turn out to belong to special classes of Boolean functions [11]. This approach is based on Boolean signals correction (BSC) in a CED circuit. Being less investigated, it allows designing much more variants of CED circuits for the same object under diagnosis than the first approach [18]. As a result, the design problem of a self-checking device becomes simpler; moreover, the designer can tune the indicators of structure redundancy, controllability, energy consumption, etc. Using BSC makes it possible to design a self-checking device even when other methods become inapplicable, e.g., the duplication method widespread in practice [19].

This paper presents new results in the field of application of BSC together with binary redundant codes. We propose a design method for self-checking combinational devices with BSC and weight-based sum codes in the ring of modulo M residues (weight-based Bose–Lin codes).

1. THE STRUCTURE OF A CED CIRCUIT WITH BOOLEAN SIGNALS CORRECTION

Figure 1 shows the structural diagram of a CED circuit based on BSC. Here, the object under diagnosis is a block $F(X)$, equipped with t inputs and n outputs. Boolean vectors $<X> = <x_t\, x_{t-1} \ldots x_2\, x_1>$ are supplied at the inputs of this block during its operation. They are to compute the values of functions implemented by the block $F(X)$ and to form a Boolean vector $<f_n(X)\, f_{n-1}(X) \ldots f_2(X)\, f_1(X)>$. The faults occurring during the operation of the block $F(X)$ distort the values of the vector $<f_n(X)\, f_{n-1}(X) \ldots f_2(X)\, f_1(X)>$ formed at its outputs. By monitoring the occurrence of these distortions, it is possible to indirectly determine the presence of faults in the object under diagnosis [1].

A CED circuit is installed to check computations at the outputs of the object of diagnosis. In contrast to the classical structure (e.g., see [1]), the CED circuit in Fig. 1 performs not the complement of the vector $<f_n(X)\, f_{n-1}(X) \ldots f_2(X)\, f_1(X)>$ but Boolean signals correction using modulo $M = 2$ addition (XOR gates). The vector $<f_n(X)\, f_{n-1}(X) \ldots f_2(X)\, f_1(X)>$ is transformed into a vector $<h_n(X)\, h_{n-1}(X) \ldots h_2(X)\, h_1(X)>$, which may have special diagnostic properties: for example, it may belong to the set of codewords of a preselected uniform binary code.

Each value $f_i(X), i=\overline{1,n},$ is corrected using two-input XOR gates: the first inputs receive the values $f_i(X)$ and the second ones the values of the same-name correction functions $g_i(X)$ of the same name. They are computed for the same sets of input variables as the values $f_i(X)$ by the correction function block $G(X)$.

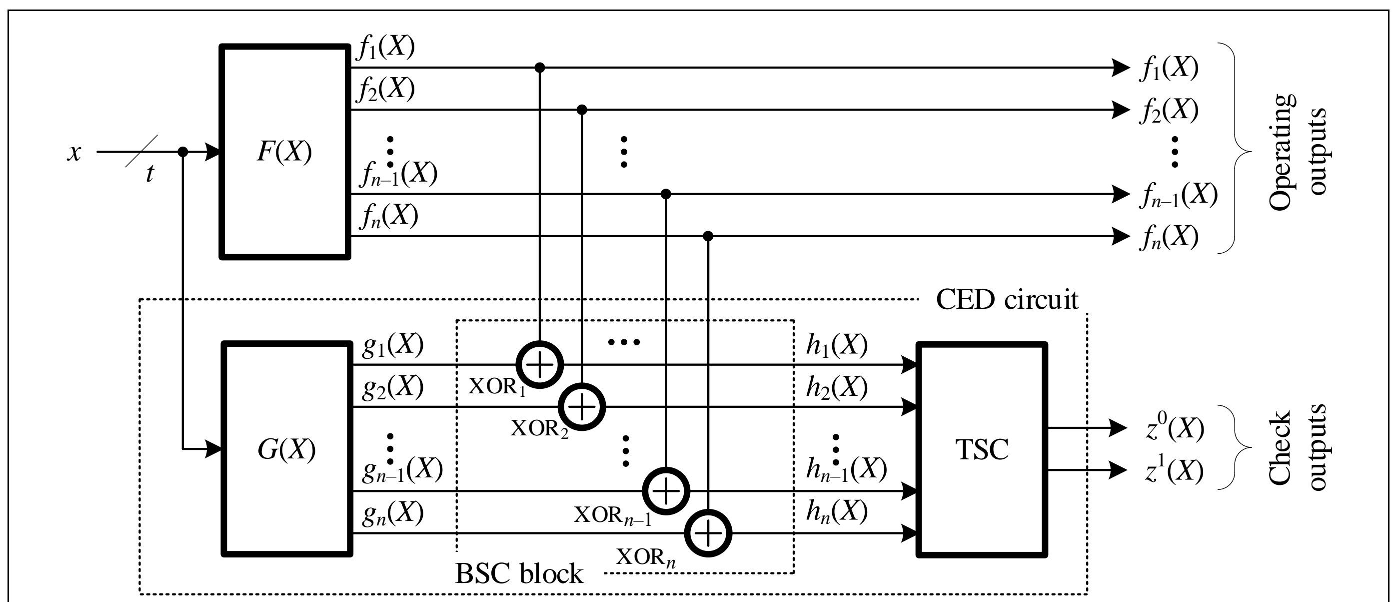

Fig. 1. The structural diagram of a CED circuit based on Boolean signals correction.

Thus, the transformations in the CED circuit are performed by the rule

$$h_i(X) = f_i(X) \oplus g_i(X), i = \overline{1, n}.$$

The cascade of XOR gates forms the Boolean signals correction block (the BSC block). A totally self-checking checker (TSC) is installed to verify whether the vector $<h_n(X)\ h_{n-1}(X)\ \ldots\ h_2(X)\ h_1(X)>$ formed at the BSC block outputs belongs to the set of codewords of the code selected for checking computations. This device has two outputs $z^0(X)$ and $z^1(X)$, which are the check outputs of the CED circuit: the presence of a two-rail signal <01> or <10> at the outputs indicates of no computational errors at the outputs of the object under diagnosis and in the CED circuit blocks; violation of the two-rail property points to the presence of errors.

The structure in Fig. 1 has an important peculiarity that many CED circuits can be built for one selected uniform binary code. This peculiarity allows the designer to solve the most difficult task of organizing a CED circuit, i.e., to ensure its self-checking structure. In addition, the designer can regulate the indicators of structure redundancy of the self-checking device. With a preselected code, an available TSC structure, and a given implementation of the block $G(X)$, the self-checking device based on the classical CED circuit [1] has a single implementation, which complicates the total self-checking of the CED circuit and, in some cases, makes it impossible. For the case of a Berger code and a repetition code underlying a redundancy system, an example demonstrating this feature of the classical CED circuit structure was provided in [18]. Thus, even duplication with comparison of computational results, a widespread method in practice, does not always yield a self-checking CED circuit due to the difficulties in ensuring the comparator's self-checking. With the CED circuit structure based on BSC, the designer can build self-checking devices even when other methods turn out inapplicable.

The key to organizing a CED circuit based on BSC is the choice of a diagnostic attribute. As noted above, it can be the belonging of the vector $<h_n(X)\ h_{n-1}(X)\ \ldots\ h_2(X)\ h_1(X)>$ formed at the BSC block outputs to a given uniform binary code. Then the properties of error detection in the codewords of this code will determine the possibilities of detecting distortions at the outputs of the object under diagnosis. Naturally, different uniform binary codes have different error detection properties by their types (by the number of combinations of distortions of zero and unit values) and multiplicities [20, 21]. The choice of an appropriate code for checking computations becomes determinant from the viewpoints of covering the errors occurring at the outputs of the object of diagnosis and ensuring the self-checking of the CED circuit.

The following conditions must be satisfied for a CED circuit to be self-checking. During the operation of a self-checking device, it is necessary to supply a given set of input signals and ensure:

- the checkability of the block $G(X)$, i.e., the possibility of manifesting any fault from a given model as distortions at its outputs for at least one set $<x_t\ x_{t-1}\ \ldots\ x_2\ x_1>$ [5];
- formation of check tests for transformation elements in the BSC block. The full test of the XOR gate in its canonical implementation contains all four combinations {00, 01, 10, 11} [22];
- formation of the check test for the TSC, which is related to the redundant code chosen at the design stage and the implementation peculiarities of the TSC structure [23].

Any uniform binary block codes (both non-separable and separable) can be chosen as base codes for designing the CED circuit shown in Fig. 1. In the case of non-separable codes, it is impossible to distinguish data and check symbols in codewords; in the case of separable codes, data symbols can be combined into the data vector and check symbols into the check vector. When choosing an appropriate code, it is reasonable to start from some threshold: the cardinality of the set of codewords of a given non-separable code or the number of check symbols of a given separable code. This threshold should not exceed the value $k = n$, which is the number of check symbols in codes with repetition, underlying the duplication structure widespread in practice. Duplication provides coverage of any error combinations at the outputs of the object under diagnosis but, at the same time, leads to significant structure redundancy: the implementation complexity indicators of a device with a CED circuit can be 3–4 (and even more) times higher than those of the object under diagnosis. For this reason, codes with low redundancy are often considered when designing CED circuits. Such codes include non-separable constant-weight codes and separable sum codes [1, 2, 11, 20, 21, 23]. Special attention is paid to the codes with redundancy close to the minimum possible for detecting errors at the outputs of objects under diagnosis. The lower bound $k = 1$ is achieved when using a parity code. Therefore, it is interesting to study the applicability of redundant parity codes with $k = 2$ (and a close number of check symbols) to checking computations.

To construct a code with $k = 2$ check symbols, we can count the number of significant signals in the data vector in the ring of modulo $M = 4$ residues. Then the so-called residue or modular sum codes will be constructed [1]. They are often termed Bose–Lin codes in

the literature [23]. For example, when designing CED circuits based on the classical structure with the complement of the data vector formed at the outputs of the object under diagnosis by the check vector, the advantages of using such codes were demonstrated in [24], including the gain compared to duplication and a Berger code applied for checking computations. Bose–Lin codes can also be used to design CED circuits based on BSC. A CED circuit design method based on BSC with such codes was presented in [25]; this method involves the transformation of only some values of the vector $<f_n(X) f_{n-1}(X) \dots f_2(X) f_1(X)>$, responsible for forming check symbols of Bose–Lin codes.

Despite their advantages of simple construction, Bose–Lin codes are not optimal from the viewpoint of error detection in data vectors with given m and k (the numbers of data and check symbols, respectively). Such are the codes where all 2^m data vectors can be uniformly distributed into groups corresponding to all 2^k check vectors [26]. There is a method for constructing some modified Bose–Lin code with modulus $M = 4$ possessing this property.

The problem of covering errors at the outputs of an object under diagnosis using uniform redundant binary codes was considered in many studies. Its solution for different codes was described, e.g., in [1, 20, 21]. Let us dwell on the solution of the most difficult problem of ensuring the self-checking of CED circuits.

2. THE "BASE" CODE FOR ORGANIZING THE CED CIRCUIT STRUCTURE

Let us modify the Bose–Lin code as follows. Fixing the value m (the number of data symbols), we form a data vector $<y_m\, y_{m-1} \dots y_2\, y_1>$ and assign to this data vector an array of weights $[w_m, w_{m-1}, \dots, w_2, w_1]$, $w_i \in \mathbb{N}$. Next, we determine the modulo $M = 4$ residue of the total weight of significant bits instead of the number of significant bits (the weight of the data vector) in the ring of modulo $M = 4$ residues:

$$W_4 = \sum_{i=1}^{m} y_i w_i \pmod 4.$$

For each data vector, the binary analog of the number W_4 is written into the bits of the check vector.

Let $WS(m, k, M)$, where $k = 2$ and $M = 4$, i.e., $WS(m, 2, 4)$, denote the modified Bose–Lin code. Consider such a code for $m = 4$.

Different combinations of weights yield a large number of $WS(4, 2, 4)$-codes. In this case, each weight can be only a number from the set {1, 2, 3} since the value of the smallest non-negative residue is written into the check vector in binary form. When setting the weights $w_i = 4j \pmod 4) = 0$, $j \in \mathbb{N}_0$, a code susceptible to interference will be constructed. (The bits whose weight is a multiple of the modulus will not be checked.)

For $WS(4, 2, 4)$-codes, there are 15 ways of constructing an interference-immune code, defined by the ways of weighting the data symbols $[w_4, w_3, w_2, w_1]$: [1, 1, 1, 1], [1, 1, 1, 2], [1, 1, 1, 3], [1, 1, 2, 2], [1, 1, 2, 3], [1, 1, 3, 3], [1, 2, 2, 2], [1, 2, 2, 3], [1, 2, 3, 3], [1, 3, 3, 3], [2, 2, 2, 2], [2, 2, 2, 3], [2, 2, 3, 3], [2, 3, 3, 3], [3, 3, 3, 3]. Note that possible permutations of weights in the arrays that do not affect the general properties of error detection in codewords are neglected here.

All $WS(4, 2, 4)$-codes with odd numbers as weights do not detect 54 errors in data vectors; $WS(4, 2, 4)$-codes with all even weights ([2, 2, 2, 2]) do not detect 112 errors in data vectors; other $WS(4, 2, 4)$-codes do not detect 48 errors in data vectors, the minimum possible number under given M and m. In all symbols of codewords, all $WS(4, 2, 4)$-codes do not detect 240 errors.

From the viewpoints of error detection in data vectors and the self-checking of test equipment for a selected separable code, it is important how many data vectors correspond to the same check vector. All data vectors can be classified into groups corresponding to the same check vector. The maximum of 2^k check vectors can be generated for a code with k check symbols. If all 2^m data vectors of the code are uniformly distributed between 2^k check vectors, such a code will detect the maximum number of errors in the data vectors. In addition, the self-checking of test equipment will be ensured much easier. For example, checkers of separable codes are most simply built in the form of two-cascade structures containing an encoder and a comparator [27]. To fully test them, it will be necessary to generate, at least once, each check vector at the comparator inputs. This is impossible if the number of check vectors of the code is not maximal and not equal to 2^k. For example, the classical Berger codes widely known have the maximum number of check vectors for k check symbols only in the case $m = 2^k - 1$. For $m \neq 2^k - 1$, the full set of check vectors is not formed for Berger codes [28]. Implementing a totally self-checking CED circuit becomes difficult. For the modified Bose–Lin codes considered in this paper, all possible check vectors are formed for k check symbols.

Consider the use of a $WS(4, 2, 4)$-code with the weights [2, 2, 2, 3] as an example. Table 1 classifies the data vectors into groups corresponding to identical check vectors for the selected $WS(4, 2, 4)$-code. According to the table, this code has the minimum possible number of undetectable errors in data vectors, and each check vector corresponds to four data vectors.

Table 1

Classification of data vectors by identical check vectors

W_4			
0	1	2	3
Check vectors			
00	01	10	11
Data vectors			
0000	0011	0010	0001
0110	0101	0100	0111
1010	1001	1000	1011
1100	1111	1110	1101

This simplifies the task of ensuring encoder testability in the CED circuit for real devices, where (as a rule) data vectors at the outputs are formed unevenly.

When constructing a *WS*(4, 2, 4)-code, one may assign other weights to obtain a different classification table of data vectors by identical check vectors. This task is not studied here. We emphasize that the encoder of the *WS*(4, 2, 4)-code under consideration, which is suitable for designing CED circuits based on BSC and contains a larger number of even weights, will have one of the simplest structures among the encoders of codes with other weights; in addition, the code will not detect the minimum possible total number of errors in data vectors.

3. THE "BASE" STRUCTURE FOR CED CIRCUIT DESIGN

We demonstrate the application of the Bose–Lin code for the structure in Fig. 1.

Figure 2 shows the "base" structure for CED circuit design for a group of six outputs of an object under diagnosis based on the *WS*(4, 2, 4)-code. Here, the values of only two functions implemented by the object are corrected in the CED circuit to form the check symbols of the code. This structure was investigated earlier, e.g., in [29], but in the case of the weights [4, 3, 2, 1]. For these weights, self-checking CED circuits can be designed only in some special cases: since the most significant bit is not controlled by check symbols, the formation of some data vectors should be excluded.

In the structure shown in Fig. 2, a *WS*(4, 2, 4)-code is used to organize checking. In the CED circuit, the codewords $<h_6(X)\ h_5(X)\ h_4(X)\ h_3(X)\ h_2(X)\ h_1(X)>$ of this code are formed at the BSC block outputs (i.e., at the TSC inputs). The two lower bits correspond to check symbols and the four higher bits to data symbols. Data symbols are formed directly at the outputs $f_3(X)$, $f_4(X)$, $f_5(X)$, and $f_6(X)$ of the object under diagnosis. The check symbols are calculated using the formulas $h_1(X)=f_1(X)\oplus g_1(X)$ and $h_2(X)=$ $=f_2(X)\oplus g_2(X)$. The functions g_1 and g_2 represent the correction functions formed by the block $G(X)$. Thus, when supplying the sets $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$ to the inputs in the CED circuit, any vector

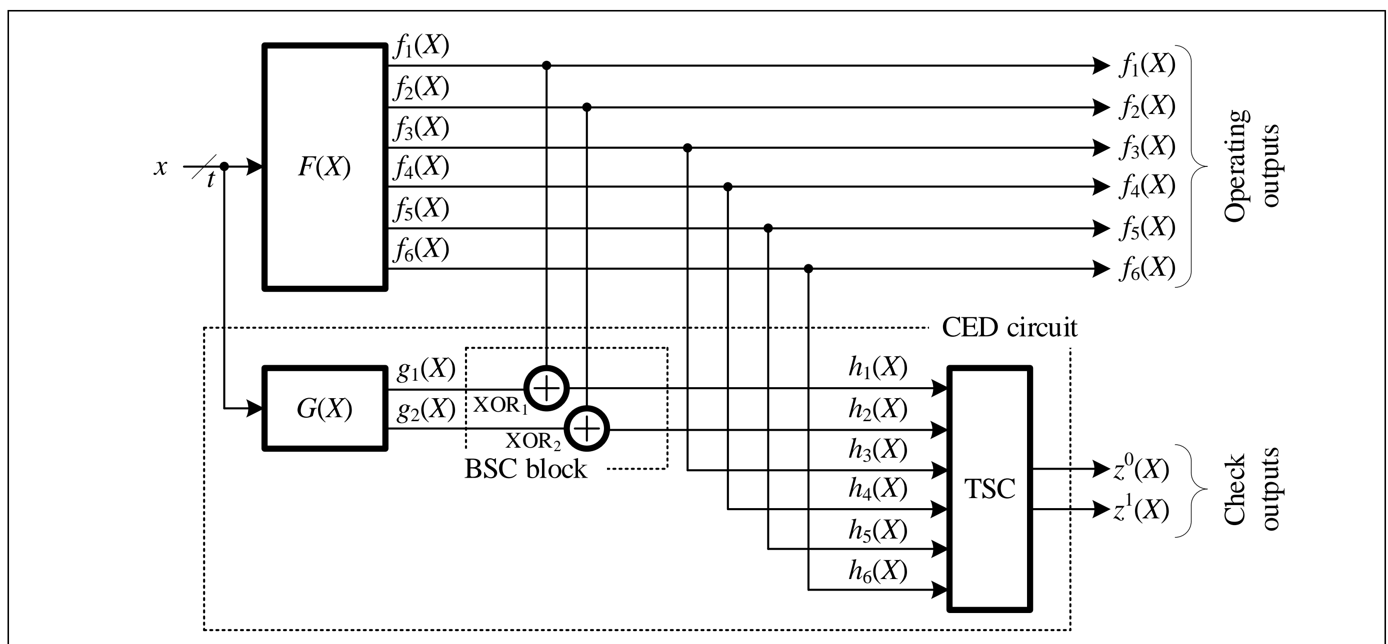

Fig. 2. The structural diagram of a CED circuit with transformation of the values of some functions implemented at the outputs of an object under diagnosis into the functions forming check symbols of the *WS*(4, 2, 4)-code.

<$f_6(X)\, f_5(X)\, f_4(X)\, f_3(X)\, f_2(X)\, f_1(X)$> is transformed into the codeword <$h_6(X)\ h_5(X)\ h_4(X)\ h_3(X)\ h_2(X)\ h_1(X)$>, which belongs to the $WS(4, 2, 4)$-code. The checker of this code has a typical structure containing an encoder and a two-rail checker (TRC) [30]. Note that the TRC block operates in the two-rail logic; therefore, it is required to pre-invert the signals from the output of the encoder of the $WS(4, 2, 4)$-code or invert the signals $h_1(X)$ and $h_2(X)$.

Let us slightly modify the structure in Fig. 2. Consider the functions implemented by the object under diagnosis and correct the values of those forming the data symbols of the $WS(4, 2, 4)$-code (Fig. 3). In this structure, only the check symbols of codewords are directly formed, and the data symbols are calculated using the block $G(X)$. Otherwise, this structure is similar to the one in Fig. 2. However, when building this structure, it is possible to ensure the self-checking of the end device by selecting the data vectors corresponding to the check vectors (see Table 1), particularly in those cases where the structure in Fig. 2 fails with this task.

The peculiarity of the modified structure is that the check vectors of the $WS(4, 2, 4)$-code in the CED circuit will be formed unambiguously, but the corresponding data vectors can be selected from four variants for each check vector (see Table 1). This peculiarity provides high variability when building the "base" structure and allows obtaining several variants of the block $G(X)$. In turn, it becomes possible to form a complete set of test combinations for XOR gates and regulate the structure redundancy indicators of the CED circuit.

Let us determine the number of ways to build the CED circuit according to the structure in Fig. 3. There are $C_{4+2}^{4} = 15$ variants for selecting the transformed outputs. There are $P_4 = 4! = 24$ variants for placing data symbols in the data vector. There are $P_2 = 2! = 2$ variants for placing check symbols in the check vector. The product of the three values mentioned characterizes the number of ways to choose the sequences of data and check symbols. Besides, on each input combination, there are exactly four ways to determine the four components of one data vector in the CED circuit: due to only 2^t input combinations, there are $4{\cdot}2^t$ ways of this determination process. The number of ways to build the CED circuit according to the "base" structure in Fig. 3 is given by

$$N_{WS(4,\,2,\,4),\,t} = 4 \cdot 2^t \cdot 15 \cdot 24 \cdot 2 = 2\,880 \cdot 2^t. \quad (1)$$

For example, for $t = 4$ inputs, we have $N_{WS(4,\,2,\,4),\,4} = 2\,880 \cdot 2^4 = 46\,080.$

In the case $t = 4$, applying the same code $WS(4, 2, 4)$ to the structure in Fig. 2 yields $N_{WS(4,\,2,\,4),\,4} = 15 \cdot 24 \cdot 2 = 720$ ways to organize the CED circuit. (It is impossible to vary the values of data symbols.)

This number is 64 times smaller than that for the structure in Fig. 3. Hence, the CED circuit structure based on BSC with the $WS(4, 2, 4)$-code proposed

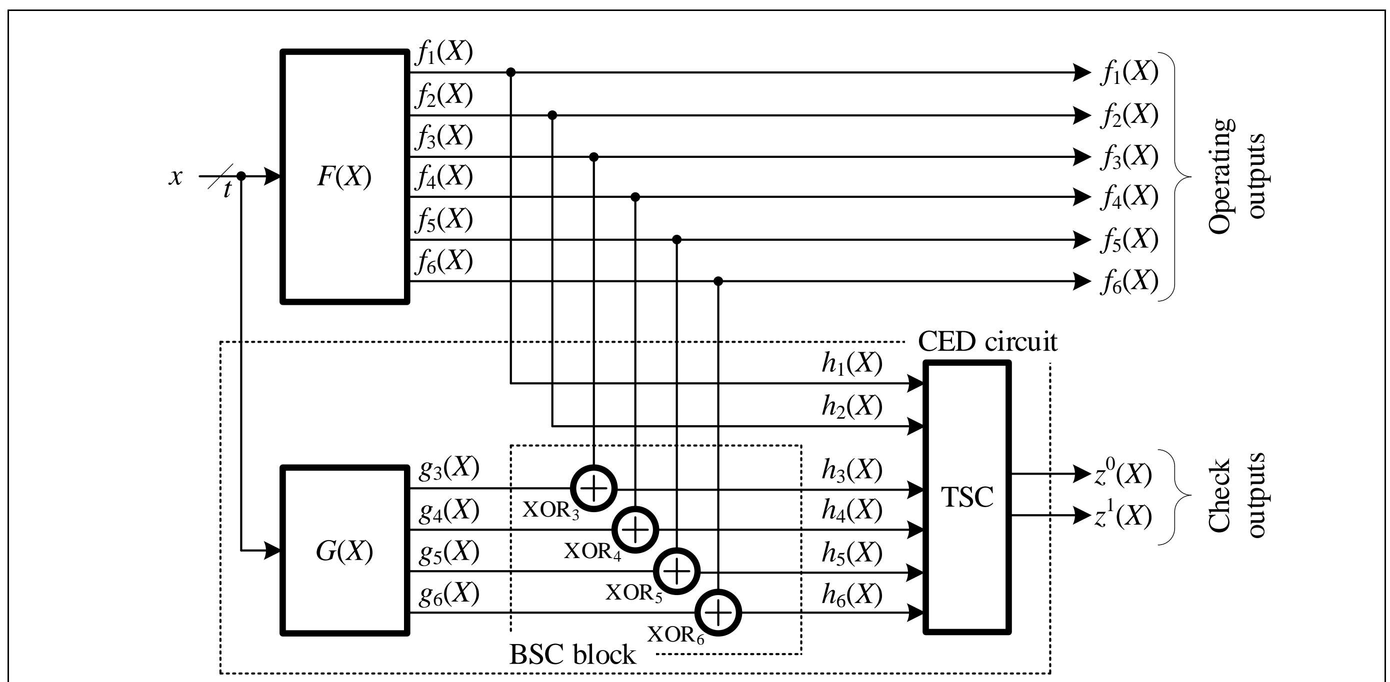

Fig. 3. The structural diagram of a CED circuit with transformation of the values of some functions implemented at the outputs of an object under diagnosis that form data symbols of the *WS*(4, 2, 4)-code.

in this paper has another advantage over the counterpart in Fig. 2, i.e., much higher variability in construction.

The "base" CED circuit structure is implemented for a six-output object under diagnosis. If an object under diagnosis has more outputs, they are divided into groups of six outputs each; for each group, a separate CED circuit is built according to the "base" structure with the subsequent union of the check outputs at the inputs of the self-checking comparator.

4. ALGORITHM FOR CED CIRCUIT DESIGN

At the design stage, it is required to ensure self-checking conditions of the CED circuit. For this purpose, the block $G(X)$ should be made checkable, which is related only to design methods: for this block, it is required to obtain a controllable structure with respect to a selected fault model that will manifest faults in the form of errors at its outputs when at least one set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$ is supplied to the inputs. It is also required to supply check tests to each element of the BSC block and the TSC. For each element of the BSC block, it is required to generate each combination from the set {00, 01, 10, 11} at least once during the operation of the self-checking device [22]. To test the checker, it is required to generate each check vector of the WS(4, 2, 4)-code at least once [20, 21]. We will implement a design algorithm for the CED circuit to ensure these conditions during the operation of the self-checking device.

The idea of the algorithm is as follows. When designing the CED circuit, it is required to unambiguously determine which codewords $<h_6(X)\ h_5(X)\ h_4(X)\ h_3(X)\ h_2(X)\ h_1(X)>$ of the WS(4, 2, 4)-code will be generated at the BSC block outputs for each set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$ arriving at the inputs. In the structure in Fig. 3, two symbols are unambiguously determined by the values $h_1(X) = f_1(X)$ and $h_2(X) = f_2(X)$, while those of the symbols $h_3(X)$, $h_4(X)$, $h_5(X)$, and $h_6(X)$ are considered indeterminate. According to Table 1, four data vectors will correspond to each check vector. Thus, at the CED circuit design stage, it is required to fix one of the four codewords of the WS(4, 2, 4)-code for each input set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$. This can be done arbitrarily. When designing the CED circuit, the main task is to determine the values of $h_3(X)$, h_4 (X), $h_5(X)$, and $h_6(X)$ at the BSC block outputs and then obtain the values of the correction functions computed by the block $G(X)$. The values of the correction functions are obtained for each input set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$ unambiguously because

$$h_i(X) = f_i(X) \oplus g_i(X) \Rightarrow g_i(X) = f_i(X) \oplus h_i(X).$$

The algorithm is intended to form the values of data symbols $h_3(X)$, h_4 (X), $h_5(X)$, and $h_6(X)$ considering the conditions of forming check tests for the elements of the BSC block and TSC.

According to the aforesaid, there are various algorithms for determining the values of the data symbols $h_3(X)$, h_4 (X), $h_5(X)$, and $h_6(X)$ at the BSC block outputs in the structure in Fig. 3 for each input set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$. Consider one of them.

A CED circuit design algorithm based on BSC using the WS(4, 2, 4)-code for a six-output device includes the following steps.

Step 1. For each function implemented at the outputs of the device $F(X)$, check whether it takes value 0 (and 1) for at least two sets of argument values. This is necessary to generate check tests for the XOR gates when determining the values of the data symbols $h_3(X)$, h_4 (X), $h_5(X)$, and $h_6(X)$ at the BSC block outputs.

Step 2. Select the outputs of the device $F(X)$ that are directly connected to the TSC and correspond to the check symbols of the WS(4, 2, 4)-code, e.g., the outputs $f_1(X)$ and $f_2(X)$. For these outputs, check the formation of each combination of values {00, 01, 10, 11} for at least one input set. This is necessary to form a check test for the TSC. If this condition fails, then other non-transformable outputs should be selected. If such a combination of two outputs does not exist, it will be impossible to build a self-checking CED circuit by the method under consideration.

Step 3. Form a value table for the outputs of the blocks $F(X)$, $G(X)$ and the BSC block, representing a truth table. This table should be filled in at the CED circuit design stage. It uniquely determines the values of $f_i(X)$ and $h_1(X) = f_1(X)$, $h_2(X) = f_2(X)$ for each input set $<x_t\ x_{t-1}\ ...\ x_2\ x_1>$. The values of the functions $h_3(X)$, $h_4(X)$, $h_5(X)$, and $h_6(X)$ and $g_3(X)$, $g_4(X)$, $g_5(X)$, and $g_6(X)$ are first considered indeterminate; they have to be obtained during the determination process.

Step 4. Fill the columns $h_3(X), h_4(X), h_5(X)$, and $h_6(X)$ row by row, starting from the column corresponding to the most significant bit of the data vector. Table 1 can be used for this purpose. For each of the functions f_5 and f_6, check whether it takes value 0 (and 1) for at least one set of the argument values from the first half (No. 0 ... $2^{t-1} - 1$), as well as for input sets from the second half (No. 2^{t-1} ... $2^t - 1$). Such a check excludes the subsequent check of the formation of all combinations from the check test for the transformation elements of the values of the functions f_5 and f_6 in the BSC block.

Step 5. For the first (second) half of the sets of argument values, assign values 0 (1, respectively) to the functions $h_6(X)$ and $h_5(X)$. As a result, the test combi-

nations <00> and <11> will be formed at least once for the first half of the input sets, and the test combinations <01> and <10> will be formed for the second half of the inputs sets for the XOR gates performing the transformations $h_6(X) = f_6(X) \oplus g_6(X)$ and $h_5(X) = f_5(X) \oplus g_5(X)$.

Step 6. Fill unambiguously the columns $h_3(X)$ and $h_4(X)$ in accordance with Table 1. Filling is unambiguous since the values of the higher bits of data vectors are determined.

Step 7. Check the condition of forming check tests for the elements performing the transformations $h_3(X) = f_3(X) \oplus g_3(X)$ and $h_4(X) = f_4(X) \oplus g_4(X)$. If the conditions are satisfied, proceed to Step 9; otherwise, change the way of filling the columns $h_5(X)$ and $h_6(X)$ by rearranging the outputs of the object under diagnosis.

Step 8. Determine the values of the functions $g_i(X) = f_i(X) \oplus h_i(X), i \in \{3, 4, 5, 6\}$.

Step 9. Optimize the functions using any known method [31].

Step 10. Design the block $G(X)$ in the selected components.

We demonstrate this algorithm on the following example.

Example. Let a combinational device be described in the first seven rows of Table 2. Here the sets of input arguments are numbered with decimals corresponding to the binary numbers formed by each set of input arguments.

At *Step 1*, we check that each function implemented at the outputs of the device $F(X)$ takes value 0 (and 1) for at least two sets of argument values. In the example under consideration, this condition is satisfied, which follows from the analysis of Table 2. Next, at *Step 2*, we determine the outputs of the object under diagnosis that are directly connected to the TSC. Assume that these are the outputs $f_1(X)$ and $f_2(X)$.

Step 3 consists in forming a value table for the outputs of the blocks $F(X)$, $G(X)$ and the BSC block (Table 2). At this step, we also unambiguously fill the values of the functions $f_i(X)$ and $h_1(X) = f_1(X)$, $h_2(X) = f_2(X)$ for each input set $<x_t\ x_{t-1}\ \ldots\ x_2\ x_1>$. The values of the functions $h_4(X)$ and $h_3(X)$ are indeterminate at this step. At *Step 4*, we check the formation of values 0 (and 1) of the functions f_5 and f_6 for at least one of the first and second halves of the sets of argument values. We fill the columns $h_6(X)$ and $h_5(X)$ at *Step 5* (Table 2). At *Step 6*, we unambiguously fill the columns $h_3(X)$ and $h_4(X)$. The filled table is presented below (see Table 3).

Table 2

Signals received when forming the values of functions h_6 and h_5

No.	$f_6(X)$	$f_5(X)$	$f_4(X)$	$f_3(X)$	$f_2(X)$	$f_1(X)$	$h_6(X)$	$h_5(X)$	$h_4(X)$	$h_3(X)$	$h_2(X)$	$h_1(X)$	$g_6(X)$	$g_5(X)$	$g_4(X)$	$g_3(X)$	Combinations at the inputs of XOR gates			
																	XOR_6	XOR_5	XOR_4	XOR_3
0	1	0	1	1	1	0	0	0	–	–	1	0	–	–	–	–	11	00	–	–
1	0	1	1	0	0	0	0	0	–	–	0	0	–	–	–	–	00	11	–	–
2	0	0	0	1	0	1	0	0	–	–	0	1	–	–	–	–	00	00	–	–
3	0	1	1	0	1	0	0	0	–	–	1	0	–	–	–	–	00	11	–	–
4	0	1	0	0	0	1	0	0	–	–	0	1	–	–	–	–	00	11	–	–
5	1	1	1	0	1	0	0	0	–	–	1	0	–	–	–	–	11	11	–	–
6	1	1	0	1	0	1	0	0	–	–	0	1	–	–	–	–	11	11	–	–
7	1	0	1	0	1	1	0	0	–	–	1	1	–	–	–	–	11	00	–	–
8	1	1	1	0	1	0	1	1	–	–	1	0	–	–	–	–	10	10	–	–
9	0	0	1	0	0	1	1	1	–	–	0	1	–	–	–	–	01	01	–	–
10	1	0	0	1	1	0	1	1	–	–	1	0	–	–	–	–	10	01	–	–
11	0	1	0	1	0	0	1	1	–	–	0	0	–	–	–	–	01	10	–	–
12	0	1	0	0	0	0	1	1	–	–	0	0	–	–	–	–	01	10	–	–
13	0	0	1	1	0	0	1	1	–	–	0	0	–	–	–	–	01	01	–	–
14	1	1	0	1	0	1	1	1	–	–	0	1	–	–	–	–	10	10	–	–
15	1	0	1	1	1	0	1	1	–	–	1	0	–	–	–	–	10	01	–	–

Note. The symbol "–" indicates the columns with indeterminate values before Step 7.

Table 3

The values of signals in the CED circuit

No.	$f_6(X)$	$f_5(X)$	$f_4(X)$	$f_3(X)$	$f_2(X)$	$f_1(X)$	$h_6(X)$	$h_5(X)$	$h_4(X)$	$h_3(X)$	$h_2(X)$	$h_1(X)$	$g_6(X)$	$g_5(X)$	$g_4(X)$	$g_3(X)$	Combinations at the inputs of XOR gates			
																	XOR_6	XOR_5	XOR_4	XOR_3
0	1	0	1	1	1	0	0	0	1	0	1	0	1	0	0	1	11	00	10	11
1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	00	11	11	00
2	0	0	0	1	0	1	0	0	1	1	0	1	0	0	1	0	00	00	01	10
3	0	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	00	11	10	00
4	0	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1	00	11	01	01
5	1	1	1	0	1	0	0	0	1	0	1	0	1	1	0	0	11	11	10	00
6	1	1	0	1	0	1	0	0	1	1	0	1	1	1	1	0	11	11	01	10
7	1	0	1	0	1	1	0	0	0	1	1	1	1	0	1	1	11	00	11	01
8	1	1	1	0	1	0	1	1	1	0	1	0	0	0	0	0	10	10	10	00
9	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	1	01	01	10	01
10	1	0	0	1	1	0	1	1	1	0	1	0	0	1	1	1	10	01	01	11
11	0	1	0	1	0	0	1	1	0	0	0	0	1	0	0	1	01	10	00	11
12	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	01	10	00	00
13	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	1	01	01	11	11
14	1	1	0	1	0	1	1	1	1	1	0	1	0	0	1	0	10	10	01	10
15	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	10	01	10	11

Next, at *Step 7*, we check the formation of check tests for the transformation elements of the values of the functions f_4 and f_3. The check tests are formed (see Table 3). At *Step 8*, we determine the values of the functions $g_i(X) = f_i(X) \oplus h_i(X), i \in \{3, 4, 5, 6\}$.

Then we optimize the functions g_3, …, g_6 (*Step 9*). Here are only the numbers of the allowed sets for the functions g_3, …, g_6: $g_6(X) = \{0, 5, 6, 7, 9, 11, 12, 13\}$, $g_5(X) = \{1, 3, 4, 5, 6, 9, 10, 13, 15\}$, $g_4(X) = \{1, 2, 4, 6, 7, 10, 13, 14\}$, $g_3(X) = \{0, 4, 7, 9, 10, 11, 13, 15\}$. (Alternatively, for each of these functions, it is possible to write all conjunctions on which they take unit values.) The subsequent optimization steps are trivial: each function can be optimized individually or as a system of Boolean functions [31]. At *Step 10*, the components are selected and the CED circuit is designed. The actions of this step need no detailed description. ♦

Now, we define the implementation complexity indicator L of the CED circuit in a preselected metric (e.g., the number of internal element inputs or the number of transistors used in the device on particular components). This indicator can be compared with that of the CED circuit designed by the duplication method, denoted by L_D. If $L < L_D$, then the method proposed in this paper is more efficient than duplication when ensuring the self-checking of both structures. Otherwise, another partitioning method is chosen, and the steps of the algorithm are repeated.

Due to formula (1), the time complexity of the proposed algorithm is asymptotically estimated by the value $2^{O(t)}$: the design problem is solved in exponential time with a linear exponent.

The presented algorithm is based on the following property of the WS(4, 2, 4)-code with the weights [2, 2, 2, 3]: for each check vector, four data vectors are formed once with the high bits 00, 01, 10, and 11, respectively. Note that WS(4, 2, 4)-codes with the weights [1, 1, 1, 2], [1, 1, 2, 3], and [1, 2, 2, 3] have the same property. These codes can also be used together with the proposed algorithm for designing the "base" structure (see Section 4). For other variants of weighting data symbols, a similar algorithm can be developed, e.g., by considering the non-repeatability of the other two data symbols in the data vectors for each check vector. For example, the WS(4, 2, 4)-codes with the weights [1, 1, 1, 2], [1, 1, 2, 2], [1, 1, 2, 3], [1, 2, 2, 2], [2, 2, 2, 3], [2, 2, 3, 3], and [2, 3, 3, 3] have a characteristic feature: for each check vector, four data vectors are formed, where each of the second and third high bits once take values 00, 01, 10, and 11, respectively.

Also, we mention an important fact: during its operation, the algorithm does not cover any error combinations at the outputs of the object under diagnosis. However, when designing a self-checking device, it is necessary to preselect implementation components, a fault model, and appropriate design methods [1, 11, 20, 21]. The problem of covering errors caused by

faults from a given model is solved by various methods, e.g., by transforming the structures into controllable ones with a given code or by selecting controllable groups of outputs [32, 33].

5. CED CIRCUIT DESIGN FOR DEVICES WITH MORE THAN SIX OUTPUTS

For multi-output devices with $n > 6$ outputs, the CED circuit is designed in accordance with the structure presented in Fig. 4. In this case, the set of outputs $W = \{f_1, f_2, ..., f_{n-1}, f_n\}$ of the source device is divided into subsets W_1, W_2, ..., W_{q-1}, W_q of cardinality 6, $q = \left\lceil \frac{n}{6} \right\rceil$. If $n(\mathrm{mod}\ 6) = 0$, all q groups will contain six non-repeating outputs each; otherwise, $(q–1)$ groups will contain six non-repeating outputs each, and the last group is formed from the outputs $W_q = \{f_{n-5}, f_{n-4}, f_{n-3}, f_{n-2}, f_{n-1}, f_n\}$. For each group of six outputs of the object under diagnosis, the structure in Fig. 3 is designed by the above algorithm. The outputs of each TSC_1, ..., TSC_q are united at the inputs of the self-checking comparator qTRC1, which consists of $(q\text{-}1)$ TRC blocks.

The resulting self-checking device can be compared in terms of implementation complexity, e.g., with the device based on duplication. For this purpose, we define the implementation complexity indicator of the self-checking device (see above) in the given metric: $L = \sum_{i=1}^{q} L_i$. Then the efficiency is evaluated in comparison with the duplication method.

Fig. 4. The structural diagram of a CED circuit for devices with $n > 6$ based on BSC with transformation of the values of some functions implemented at the outputs of an object under diagnosis that form data symbols of the $WS(4, 2, 4)$-code.

Note that there are many ways to select groups of outputs into subsets of cardinality 6. The outputs for the set W_1 can be selected in C_n^6 ways. The outputs for the set W_2 can be selected in C_{n-6}^6 ways, and so on. Thus, for an arbitrary $j \in \{1, q-1\}$, we have $C_{n-6(j-1)}^6$ possible ways. (The set W_q is chosen uniquely.) The total number of ways to build the CED circuit is $\prod_{j=1}^{q-1} C_{n-6(j-1)}^6$. This value can be rewritten as

$$\prod_{j=1}^{q-1} C_{n-6(j-1)}^6 = C_n^6 C_{n-6}^6 C_{n-12}^6 \cdot \ldots \cdot C_{n-6(q-3)}^6 C_{n-6(q-2)}^6$$

$$= \frac{n!}{6!(n-6)!} \cdot \frac{(n-6)!}{6!(n-12)!} \cdot \frac{(n-12)!}{6!(n-18)!} \cdot \ldots \cdot \frac{(n-6(q-3))!}{6!(n-6(q-2))!} \cdot \frac{(n-6(q-2))!}{6!(n-6(q-1))!} \quad (2)$$

$$= \frac{n!}{(6!)^{q-1}(n-6q+6)!}.$$

Since $q = \left\lceil \frac{n}{6} \right\rceil$, the expression (2) can be represented as a function of the number n only:

$$\prod_{j=1}^{\left\lceil \frac{n}{6} \right\rceil - 1} C_{n-6(j-1)}^6 = \frac{n!}{(6!)^{\left\lceil \frac{n}{6} \right\rceil - 1} \left(n - 6\left\lceil \frac{n}{6} \right\rceil + 6 \right)!}. \quad (3)$$

For example, for a device with $n = 20$ outputs, formula (3) gives the following number of ways to select the subsets:

$$\prod_{j=1}^{\left\lceil \frac{20}{6} \right\rceil - 1 = 3} C_{20-6(j-1)}^6 = \frac{20!}{(6!)^{\left\lceil \frac{20}{6} \right\rceil - 1} \left(20 - 6\left\lceil \frac{20}{6} \right\rceil + 6 \right)!}$$

$$= \frac{20!}{(6!)^3 \, 2!} = 3\,259\,095\,840.$$

These formulas indicate the total number of ways to select the subsets of outputs considering the minimum number of groups required for complete coverage without selecting the same check outputs in different groups (possibly, except for the last group). Including the same outputs in different groups for checking may be necessary to detect the required combinations of distortions in the outputs. Anyway, due to this circumstance, the number of ways to organize the CED circuit can be increased in comparison with the one yielded by the above algorithm and formula (3).

Concluding this section, we draw the reader's attention to the fact that, by the expression (3), all groups are analyzed in factorial time. (In other words, the time complexity is estimated by the value $O(n!)$.)

6. RESULTS OF EXPERIMENTS WITH TEST COMBINATIONAL CIRCUITS

Experiments with test combinational circuits from MCNC Benchmarks [34] were carried out to verify the efficiency of the proposed CED circuit design method based on BSC with $WS(4, 2, 4)$-codes. For each benchmark, the implementation complexity indicators were experimentally estimated for the devices with the CED circuit designed in accordance with the structure in Fig. 4. The SIS interpreter [35, 36] was used to optimize the logic correction functions, and the block $G(X)$ was designed. The implementation complexity indicators were determined in conventional units of the stdcell2_2.genlib library (the area occupied by the device on a chip). Some experimental results are summarized in Table 4 and additionally presented in Fig. 5. For each benchmark, the table provides the values of the corresponding parameters (t and n, the numbers of inputs and outputs), the number of selected groups (q), as well as the values of the implementation complexity indicators ($L_{F(X)}$, $L_{G(X)}$, and L, the conditional implementation complexity indicators of the blocks $F(X)$, $G(X)$ and the device with the CED circuit, respectively). For comparison, the column L_D presents the value of the implementation complexity indicator of the device designed by the duplication method; the last column of the table, the share δ of the implementation complexity indicator of the device with the CED circuit designed by the presented method in the corresponding indicator of the device with the CED circuit designed by the duplication method:

$$\delta = \frac{L}{L_D} \cdot 100\%.$$

No permutations of the outputs of the benchmarks were performed during the experiments. The groups of outputs were selected in their order in the description of the benchmark. For each group of outputs, the CED circuit was built, the blocks $G_j(X)$, $j = \overline{1, q}$, were jointly implemented, and the structure redundancy indicators were estimated.

The following results were obtained for the 20 benchmarks presented in Table 4 and Fig. 5. In 18 cases, the structure redundancy indicator decreased in comparison with duplication. The value of $\delta = 81.729\%$ was calculated on the average for the 20 benchmarks considered. Therefore, the proposed CED

circuit design method has high efficiency in terms of structure redundancy.

For some benchmarks, arbitrary selection of groups of outputs without any output permutations within the groups gave no possibility to implement a self-checking CED circuit with forming all test combinations for the transformation elements and checker. However, simple permutation procedures for outputs within groups provide an effective solution. Here we describe the results for the dc1 circuit.

Initially, the outputs of the benchmarks were not permuted. For the dc1 circuit, the resulting distribution of test combinations for the CED circuit elements is presented in Table 5. Obviously, the number of test combinations has an uneven distribution, which is certainly due to the peculiarities of the dc1 circuit. (Besides the various arrangements of zero and one at the outputs, this circuit has zero at all outputs for the input combinations <1010> ... <1111>.) Also, the test combination <01> is not formed for the XOR_4 gate. With a simple permutation of the outputs f_5 and f_4 without changing the way to build the CED, the test combinations were redistributed, and the combination <01> was formed for the element XOR_4 in the CED circuit for the first group of outputs. The number of test combinations for TSC_1 and TSC_2 in both cases remained the same since only two outputs were permuted before the determination stage of the codewords of the *WS*(4, 2, 4)-code.

The output permutation also affected the structure redundancy of the self-checking device. In the first (original) case, the joint implementation complexity indicator of the blocks $G_j(X)$ was equal to 696 conditional units of the stdcell2_2.genlib library; in the second case, this indicator became equal to 672 conditional units of the stdcell2_2.genlib library. This decreased the value of L from 2872 to 2848 and the value of δ from 89.303% to 88.557% for this CED circuit design. This decrease is not significant.

Due to a large number of possible output permutations and ways to select controlled groups, one can form check tests for all elements of the CED circuit and regulate the structure redundancy indicators of the self-checking device.

Table 4

Experimental results

No.	Benchmark	t	n	q	$L_{F(X)}$, cond. units	$L_{G(X)}$, cond. units	L, cond. units	L_D, cond. units	δ, %
1	dc1	4	7	2	976	696	2872	3216	89.303
2	dekoder	4	7	2	736	752	2688	2736	98.246
3	wim	4	7	2	712	656	2568	2688	95.536
4	newbyte	5	8	2	592	680	2472	2656	93.072
5	p82	5	14	3	2368	1712	5976	7456	80.15
6	m1	6	12	2	3064	880	5144	8432	61.006
7	newapla2	6	7	2	600	592	2392	2464	97.078
8	sqr6	6	12	2	2648	2184	6032	7600	79.368
9	inc	7	9	2	2376	1792	5368	6432	83.458
10	newcpla2	7	10	2	1896	1440	4536	5680	79.859
11	max128	7	24	4	20 192	2520	25 304	45 184	56.002
12	m2	8	16	3	10 096	3024	15 016	23 328	64.369
13	m3	8	16	3	13 464	3744	19 104	30 064	63.544
14	m4	8	16	3	18 704	7152	27 752	40 544	68.449
15	mlp4	8	8	2	7224	9224	17648	15 920	110.854
16	tms	8	16	3	6784	3032	11 712	16 704	70.115
17	dk27	9	9	2	528	1168	2896	2736	105.848
18	max512	9	6	1	9632	5624	15 760	20 320	77.559
19	newcpla1	9	16	3	2520	2528	6944	8176	84.932
20	newxcpla1	9	23	4	3760	2832	9184	12 112	75.826
Average value									81.729

Table 5

The number of test combinations for CED circuit elements for the dc1 circuit

Group number	Gate	Variant I				Variant II			
		00	01	10	11	00	01	10	11
1	XOR_6	5	7	1	3	5	7	1	3
	XOR_5	3	6	2	5	3	6	2	5
	XOR_4	9	0	3	4	8	1	2	5
	XOR_3	6	4	5	1	6	4	5	1
	TSC_1	6	2	1	7	6	2	1	7
2	XOR_7	4	6	2	4	4	6	2	4
	XOR_6	5	7	1	3	5	7	1	3
	XOR_5	6	3	3	4	7	2	4	3
	XOR_4	6	3	5	2	7	2	6	1
	TSC_2	6	4	2	4	6	4	2	4

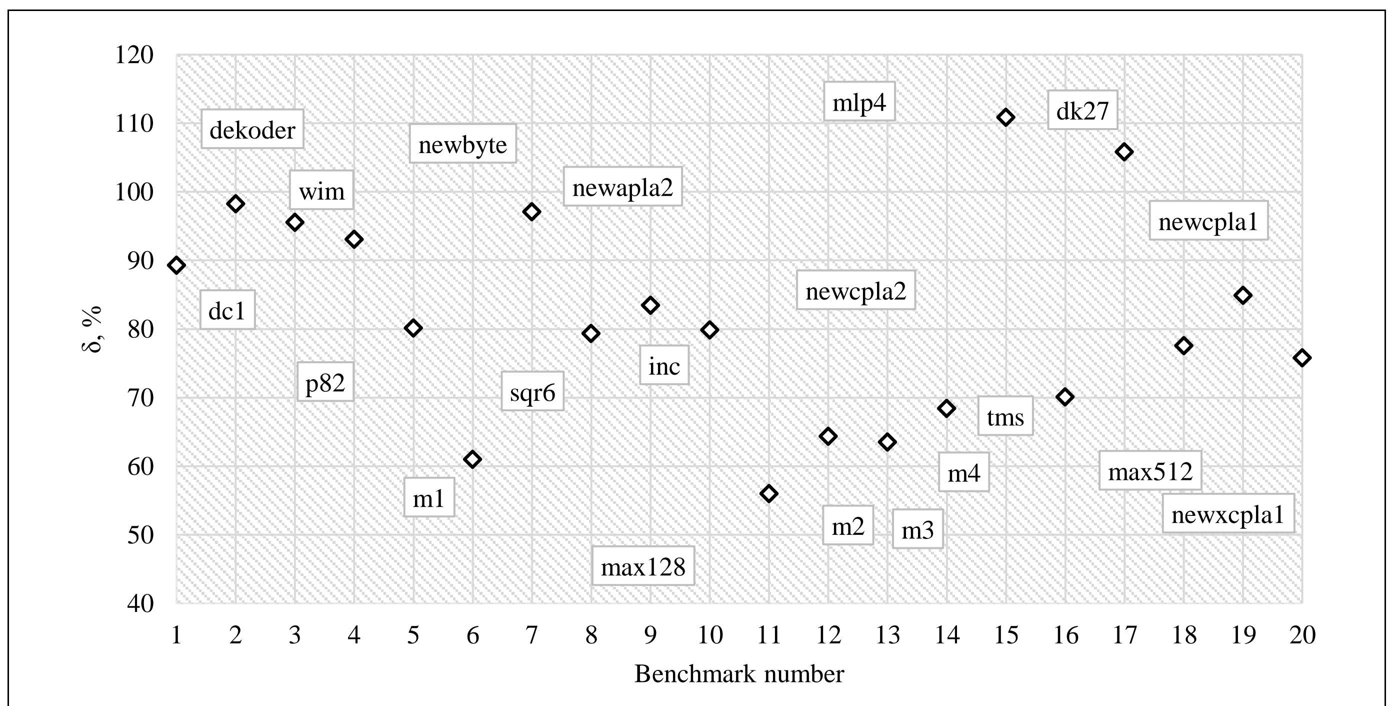

Fig. 5. The graphical representation of the experimental results.

Further studies and experiments may aim to minimize the value of the indicator L, to ensure the checkability of all CED circuit elements, and to obtain a uniform distribution of test combinations for encoders and transformation elements in CED circuit structures. These tasks go beyond the scope of the paper.

CONCLUSIONS

The CED circuit design method with BSC using $WS(4, 2, 4)$-codes and the formation of the "base" structure of a self-checking device, the method proposed in this paper, provides the designer of such devices with higher variability compared to the method known previously, in which correction applies in the CED circuit only to the functions participating in the formation of check code symbols. In turn, the designer can optimize the indicators of structure redundancy, controllability, energy consumption, etc. for the self-checking devices being built.

According to the experimental results presented above, in many cases it is possible to reduce the structure redundancy indicators in comparison with the standard duplication method; for about half of the cir-

cuits, it is possible to achieve δ = 50–70%. Therefore, the method can be applied in practice to implement self-checking digital devices.

The method has the following advantages: if there is a possibility to build a self-checking device by this method, it will yield results. This circumstance follows from the need to check, for each function implemented at the outputs of the device $F(X)$, whether it takes value 0 (and 1) for at least two sets of argument values. If this condition fails, a self-checking device will not be designed even using standard methods (e.g., duplication): a check test for the comparator will not be formed. In addition, the method surely forms test combinations for the first two transformation elements out of four. Among the drawbacks of the method, we note that all combinations included in the check test are not necessarily formed for the remaining two transformation elements; see the experimental results for several test circuits. In this case, it is necessary to permute the outputs and, possibly, to regroup them.

Also, we emphasize that the proposed "base" structure yields a large number of variants to form the values of correction functions whereas the presented algorithm only one of a few. This fact makes application of the method promising, particularly with other algorithms for designing self-checking devices.

An interesting development of the method presented above is a generalized CED circuit design method with BSC using weight-based Bose–Lin codes with an arbitrary value of m and different values of the moduli $M \in \left\{2^2, 2^3, ..., 2^{\lceil \log_2(m+1) \rceil}\right\}$. There are at least two lines of related research. The first one is to study the peculiarities of applying a sequence of natural number series weights when constructing the code: the presence of weights representing multiples of the modulus value leads to the appearance and, as m grows, to an increase in the number of single undetectable errors [21]. The second line is to study the peculiarities of applying sequences of arbitrary natural number weights.

Note finally that the use of BSC in the design of self-checking CED circuits is an unexplored approach to building devices and systems with fault detection, which may significantly improve their efficiency compared to known methods.

REFERENCES

1. Sogomonyan, E.S. and Slabakov, E.V., *Samoproveryaemye ustroistva i otkazoustoichivye sistemy* (Self-Checking Devices and Fault-Tolerant Systems), Moscow: Radio i Svyaz', 1989. (In Russian.)
2. Lala, P.K., *Self-Checking and Fault-Tolerant Digital Design*, San Francisco: Morgan Kaufmann Publishers, 2001.
3. Gharibi, W., Hahanov, V., Chumachenko, S., et al., Vector-Logic Computing for Faults-As-Address Deductive Simulation, *IAES International Journal of Robotics and Automation*, 2023, vol. 12, no. 3, pp. 274–288. DOI: 10.11591/ijra.v12i3.pp274-288.
4. Ubar, R., Raik, J., Jenihhin, M., and Jutman, A., *Structural Decision Diagrams in Digital Test: Theory and Applications*, Cham: Springer Nature, 2024. DOI: 10.1007/978-3-031-44734-1.
5. Parkhomenko, P.P. and Sogomonyan, E.S., *Osnovy tekhnicheskoi diagnostiki. Tom 2: Optimizatsiya algoritmov diagnostirovaniya, apparaturnye sredstva)* (Foundations of Technical Diagnosis. Vol. 2: Optimization of Diagnostic Algorithms, Hardware Means), Moscow: Energoatomizdat, 1981. (In Russian.)
6. Drozd, A.V., Kharchenko, V.S., and Antoshchuk, S.G., *Rabochee diagnostirovanie bezopasnykh informatsionno-upravlyayushchikh sistem* (Operational Diagnosis of Safe Information and Control Systems), Khar'kov: Zhukovskii National Aerospace University, 2012. (In Russian.)
7. Mikoni, S., Top Level Diagnostic Models of Complex Objects, in *Lecture Notes in Networks and Systems*, 2022, vol. 442, pp. 238–249. DOI: 10.1007/978-3-030-98832-6_21.
8. Bennetts, R.G., *Design of Testable Logic Circuits*, Boston: Addison-Wesley, 1984.
9. McCluskey, E.J., *Logic Design Principles (with Emphasis on Testable Semicustom Circuits)*, New Jersey: Prentice-Hall, 1986.
10. Abramovici, M., Breuer, M.A., and Friedman, A.D., *Digital System Testing and Testable Design*, New Jersey: IEEE Press, 1998.
11. Göessel, M., Ocheretny, V., Sogomonyan, E., and Marienfeld, D., *New Methods of Concurrent Checking*, 1st ed., Dordrecht: Springer Science+Business Media, 2008.
12. Sahana, A.R., Chiraag, V., Suresh, G., et al., Application of Error Detection and Correction Techniques to Self-Checking VLSI Systems: An Overview, *Proceedings of 2023 IEEE Guwahati Subsection Conference (GCON)*, Guwahati, 2023. DOI: 10.1109/GCON58516.2023.10183449.
13. Mitra, S. and McCluskey, E.J., Which Concurrent Error Detection Scheme to Choose?, *Proceedings of International Test Conference*, Atlantic City, 2000, pp. 985–994. DOI: 10.1109/TEST.2000.894311.
14. Sagalovich, Yu.L., and Solomennikov, V.Yu., Fault Detection in Network Realizations of Systems of Monotone Boolean Functions, *Problems Inform. Transmission*, 1997, vol. 33, no. 2, pp. 163–173.
15. Gessel', M., Dmitriev, A.V., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., A Functional Fault-Detection Self-test for Combinational Circuits, *Automation and Remote Control*, 1999, vol. 60, no. 11, pp. 1653–1663.
16. Efanov, D.V. and Pogodina, T.S., Properties Investigation of Self-Dual Combinational Devices with Calculation Control Based on Hamming Codes, *Informatics and Automation*, 2023, vol. 22, no. 2, pp. 349–392. DOI: 10.15622/ia.22.2.5. (In Russian.)
17. Efanov, D.V., The Self-Checking Structures Implementation Features Based on the Inverting Data and Linear Block Code Method, *Tomsk State University Journal of Control and Computer Science*, 2023, no. 65, pp. 126–138. DOI: 10.17223/19988605/65/13. (In Russian.)
18. Goessel', M., Morozov, A.V., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., Checking Combinational Circuits by the

Method of Logic Complement, *Automation and Remote Control*, 2005, vol. 66, no. 8, pp. 1336–1346.
19. Goessel, M. and Graf, S., *Error Detection Circuits*, London: McGraw-Hill, 1994.
20. Sapozhnikov, V.V., Sapozhnikov, Vl.V., and Efanov, D.V., *Kody s summirovaniem dlya sistem tekhnicheskogo diagnostirovaniya. Tom 1: Klassicheskie kody Bergera i ikh modifikatsii* (Sum Codes for Technical Diagnosis Systems. Vol. 1: Classical Berger Codes and Their Modifications), Moscow: Nauka, 2020. (In Russian.)
21. Sapozhnikov, V.V., Sapozhnikov, Vl.V., and Efanov, D.V., *Kody s summirovaniem dlya sistem tekhnicheskogo diagnostirovaniya. Tom 2: Vzveshennye kody s summirovaniem*, (Sum Codes for Technical Diagnosis Systems. Vol. 2: Weight-Based Codes), Moscow: Nauka, 2021. (In Russian.)
22. Aksenova, G.P., Necessary and Sufficient Conditions for Design of Completely Checkable Modulo 2 Convolution Circuits, *Automation and Remote Control*, 1979, vol. 40, no. 9, pp. 1362–1369.
23. Piestrak, S.J., *Design of Self-Testing Checkers for Unidirectional Error Detecting Codes*, Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 1995.
24. Das, D. and Touba, N.A., Synthesis of Circuits with Low-Cost Concurrent Error Detection Based on Bose–Lin Codes, *Journal of Electronic Testing: Theory and Applications*, 1999, vol. 15, no. 1-2, pp. 145–155. DOI: 10.1023/A:1008344603814.
25. Efanov, D.V., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., The Self-Checking Concurrent Error-Detection Systems Synthesis Based on the Boolean Complement to the Bose–Lin Codes with the Modulo Value M = 4, *Electronic Modeling*, 2021, vol. 43, no. 1, pp. 28–45. DOI: 10.15407/emodel.43.01.028.
26. Efanov, D.V., Sapozhnikov, V.V., and Sapozhnikov, V.V., Sum Code Family Formation Method with Undetectable Error Minimum in Data Vectors, *Informatics*, 2019, vol. 16, no. 3, pp. 101–118. (In Russian.)
27. Sapozhnikov, V.V. and Sapozhnikov, Vl.V., *Samoproveryaemye diskretnye ustroistva* (Self-Checking Discrete Devices), St. Petersburg: Energoatomizdat, 1992. (In Russian.)
28. Efanov, D.V., Sapozhnikov, V.V., Sapozhnikov, and Vl.V., On Summation Code Properties in Functional Control Circuits, *Automation and Remote Control*, 2010, vol. 71, no. 6, pp. 1117–1123.
29. Pashukov, A.V., Application of Weight-Based Sum Codes at the Synthesis of Circuits for Built-in Control by Boolean Complement Method, *Transport Automation Research*, 2022, no. 1. pp. 101–114. DOI: https://doi.org/10.20295/2412-9186-2022-8-1-101-114. (In Russian.)
30. Carter, W.C., Duke, K.A., and Schneider, P.R., US Patent 3559167A, 1971.
31. Zakrevskii, A.D., Pottosin, Yu.V., and Cheremisinova, L.D., *Logicheskie osnovy proektirovaniya diskretnykh ustroistv* (Logical Foundations of Designing Discrete Devices), Moscow: Fizmatlit, 2007. (In Russian.)
32. Morosow, A., Sapozhnikov, V.V., Sapozhnikov, Vl.V., and Goessel, M., Self-Checking Combinational Circuits with Unidirectionally Independent Outputs, *VLSI Design*, 1998, vol. 5, no. 4, pp. 333–345. DOI: 10.1155/1998/20389.
33. Efanov, D.V., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., Organization of a Fully Self-Checking Structure of a Combinational Device Based on Searching for Groups of Symmetrically Independent Outputs, *Automatic Control and Computer Sciences*, 2020, vol. 54, no. 4, pp. 279–290. DOI: 10.3103/S0146411620040045.
34. *Collection of Digital Design Benchmarks*. URL: https://ddd.fit.cvut.cz/www/prj/Benchmarks/. (Accessed February 24, 2024.)
35. Sentovich, E.M., Singh, K.J., and Moon, C., Sequential Circuit Design Using Synthesis and Optimization, *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers & Processors*, Cambridge, 1992, pp. 328–333. DOI: 10.1109/ICCD.1992.276282.
36. Sentovich, E.M., Singh, K.J., and Lavagno, L., *SIS: A System for Sequential Circuit Synthesis*, Berkeley: Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, 1992.

This paper was recommended for publication by V.G. Lebedev, a member of the Editorial Board.

Received March 15, 2024,
and revised August 24, 2024.
Accepted August 29, 2024.

Author information

Efanov, Dmitry Viktorovich. Dr. Sci. (Eng.), Peter the Great Saint Petersburg Polytechnic University, St. Petersburg, Russia; Russian University of Transport, Moscow, Russia
✉ TrES-4b@yandex.ru
ORCID iD: https://orcid.org/0000-0002-4563-6411

Yelina, Yeseniya Igorevna. Postgraduate, Peter the Great Saint Petersburg Polytechnic University, St. Petersburg, Russia
✉ eseniya-elina@mail.ru
ORCID iD: https://orcid.org/0009-0004-4167-3591

Cite this paper

Efanov, D.V. and Yelina, Y.I., Design of Self-Checking Digital Devices with Boolean Signals Correction Using Weight-Based Bose–Lin Codes. *Control Sciences* **4**, 22–36 (2024). http://doi.org/10.25728/cs.2024.4.3

Original Russian Text © Efanov, D.V., Yelina, Y.I., 2024, published in *Problemy Upravleniya*, 2024, no. 4, pp. 26–43.

This paper is available under the Creative Commons Attribution 4.0 Worldwide License.

Translated into English by *Alexander Yu. Mazurov*,
Cand. Sci. (Phys.–Math.),
Trapeznikov Institute of Control Sciences,
Russian Academy of Sciences, Moscow, Russia
✉ alexander.mazurov08@gmail.com