ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2025, Vol. 86, No. 5, pp. 402–416.

CONTROL IN TECHNICAL SYSTEMS

Design of Self-Checking Discrete Devices Based on Polynomial Codes with Computation Control via Several Diagnostic Attributes

D. V. Efanov[*,**,a] **and D. V. Pivovarov**[***,b]

[*] *Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russia*
[**] *Russian University of Transport (MIIT), Moscow, Russia*
[***] *Emperor Alexander I St. Petersburg State Transport University, St. Petersburg, Russia*
e-mail: [a] *TrES-4b@yandex.ru,* [b] *pivovarov.d.v.spb@gmail.com*

Received November 28, 2024
Revised March 17, 2025
Accepted March 27, 2025

Abstract—It is proposed to use polynomial codes in the design of self-checking discrete devices with computation control via several diagnostic attributes. A fast algorithm is developed to obtain functions describing check symbols of polynomial codes in the form of logical expressions. As is shown, encoders of polynomial codes can be referred to devices of three types: 1) those with only self-dual Boolean functions realized at the outputs, 2) those with only nearly self-dual (self-quasidual) Boolean functions realized at the outputs, and 3) those with both self-dual and self-quasidual Boolean functions realized at the outputs. A classification of polynomial codes considering this feature is presented. The structural diagram of computation control at the outputs of self-dual discrete devices via several diagnostic attributes is described. A novel algorithm is proposed to design a fully self-checking discrete device with computation control via several diagnostic attributes. In contrast to the well-known ones, this algorithm considers the nature of errors occurring at the outputs of discrete devices and their preliminary detection by checkers of self-dual and/or self-quasidual signals. The results can be used in the development of automated design tools for discrete devices for a wide range of applications.

Keywords: self-checking discrete devices, computation control via several diagnostic attributes, control of self-duality of signals, control of self-quasiduality of signals, polynomial codes

DOI: 10.31857/S0005117925050038

1. INTRODUCTION

The growing complexity of the developed and implemented technical systems that support critical industrial processes, as well as the increase in their performance and the appearance of advanced functionality, including artificial intelligence, require special attention to the reliability and safety of operation. In this regard, the most important aspect is to provide devices with the ability to identify faults during operation.

When designing discrete systems and control devices, their structures are endowed with various properties to easily ensure the detection of occurring faults. There exist methods for designing controllable devices [1] and developing self-checking structures for them [2]. Such methods apply to any critical-purpose systems, including those with rarely changing input data (e.g., control systems in the nuclear industry, electrical interlocking in railway transport, air defense systems, etc.). In such systems, hidden faults and the accumulation of errors are not allowed: one way or another, they will ultimately violate the readiness property of the control algorithm [3, 4].

The self-checking implementation of discrete devices is required to detect their faults in due time. This is done by various methods with the introduction of structural redundancy according to definite algorithms, e.g., state coding and the use of self-checking concurrent error-detection (CED) circuits [5].

An effective approach to implementing self-checking CED circuits involves the so-called data inversion: all input and output signals are represented as sequences $0101\ldots01$ and $1010\ldots10$ instead of constant values 0 and 1 [6]. This approach to implementing self-checking discrete devices is associated with using hardware redundancy and, moreover, temporal redundancy. It includes computation control via a definite diagnostic attribute, considering the constant change of input signals and the supply of input argument sets in pairs, i.e., the operating and check sets of argument values. Self-dual functions and Boolean functions [7], "close" to the former, possess the peculiarities that can be considered when organizing diagnostic support in this operation mode. The issues of implementing self-dual discrete devices were studied in several research works, e.g., [6, 8–11]. Also, note three monographs covering the basic results from the theory of self-dual discrete device design [12–14].

Computation control via the self-duality attribute requires the self-dual implementation of discrete devices, implying an appropriate structure to describe all their outputs by self-dual functions. Not all discrete devices are self-dual. Here are some examples: a full adder, a majority element, encoders of some separable block codes, etc. [6]. However, since the number of self-dual functions of t variables equals 2^{2^t-1}, it is possible to design $C^m_{2^{2^t-1}}$ combinational devices with $m \leqslant 2^{2^t-1}$ outputs. For example, for $t = 4$ and $m = 4$, the number of different self-dual combinational discrete devices reaches 174 792 640. In addition, any structure of a combinational discrete device can be transformed into a self-dual one using only a single variable based on the well-known Shannon decomposition [15]. For a self-dual implementation of memory devices, it suffices to make the combinational parts self-dual [6]. The design peculiarities of self-dual discrete devices are well-studied; for example, see [9].

In this paper, the reader's attention is focused on the issues of organizing computation control at the outputs of self-dual discrete devices, using not only the diagnostic attribute of signal self-duality but also additional attributes. As shown by numerous experiments, this self-checking device design approach allows increasing checkability (in terms of error observability) under a smaller level of hardware redundancy compared to the well-known duplication method [16]. This effect is achieved by using properties of some noise-immune codes with self-dual and "close" functions describing their check symbols [17]. As such, we consider a wide class of polynomial, or algebraic, codes [18].

2. POLYNOMIAL CODES AND THEIR PECULIARITIES

2.1. Construction Principles of Polynomial Codes

Polynomial codes are based on the division, with remainder (residue), of binary polynomials corresponding to data messages. When dividing different polynomials of degree q by a generating polynomial $G(y)$, 2^q residues are obtained. Each check symbol of the polynomial code is thus described by a logical expression containing only mod $M = 2$ addition (the XOR operation) with the values of definite data symbols. For the sake of simplicity, let the functions describing the check symbols of the codewords of polynomial codes be called *check functions*.

When forming the codewords of polynomial codes, each bit of the code vector is assigned a variable y with degree j, which corresponds to the bit location: the least significant bit is assigned the value $j = 0$, the next the value $j = 1$, and so on. The algebraic representation of the code vector is obtained by multiplying the value of y^j by the value of the corresponding bit. For example, the codeword $< 1011 >$ can be written as a polynomial as follows: $1 \times y^3 + 0 \times y^2 + 1 \times y^1 + 1 \times y^0$. By removing the zero terms, we have the polynomial $y^3 + y + 1$.

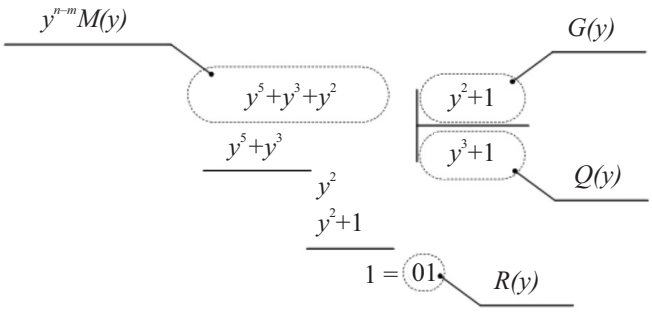

Fig. 1. The division of polynomials with residue: an illustrative example.

To form the codeword $V(y)$ of a polynomial code, the residual polynomial $R(y)$ is determined by dividing the polynomial $M(y)$, with the factor y^{n-m}, by the generating polynomial [18]: $y^{n-m}M(y) = G(y)Q(y)+R(y)$, where $Q(y)$ and $R(y)$ denote the quotient and residue from dividing $y^{n-m}M(y)$ by $G(y)$. In this case, the residue $R(y)$ will be the check vector occupying the k least significant bits of the codeword of length n. The m most significant bits will correspond to the data vector.

We encode the data vector $< 1011 >$ by a polynomial code with the generating polynomial $y^2 + 1$. This polynomial has degree 2. (It must not exceed the degree of the data polynomial.) When multiplication is performed, the polynomial $y^{n-m}M(y)$ takes the form $y^2(y^3+y+1) = y^5+y^3+y^2$.

Let us divide it by the generating polynomial $y^2 + 1$ (Fig. 1).

The polynomial corresponding to the codeword is obtained by summing the polynomials $y^{n-m}M(y)$ and $R(y)$: $y^5 + y^3 + y^2 + 1$. Thus, the encoded message becomes $< 101101 >$ (instead of the original combination $< 1011 >$). In fact, the data vector has been supplemented by the check vector $< 01 >$.

Since a generating polynomial can be mapped into a binary number, we will use the decimal equivalent N of this binary number to indicate the generating polynomial. Some error detection properties in CED circuits based on polynomial codes with different values of N were investigated in [19, 20]. Note that the generating polynomial $G(y)$ is chosen based on the condition of maximum error detection. For example, there is no sense in choosing y^0 as a generating polynomial: when dividing any polynomial of arbitrary degree by this polynomial, the residue will always be 0. (Also, zero residue is obtained when dividing by several polynomials even of the highest degree, e.g., y^2 or y^3.) Such polynomials allow detecting no errors in the code vector. We exclude from further analysis the generating polynomials without the free term y^0 (the numbers N corresponding to such polynomials are even). Therefore, we will consider polynomial codes whose generating polynomials correspond to odd numbers N. Only for such polynomials is it possible to obtain a complete set of check vectors with k bits and, accordingly, easier to provide the full self-checking of encoders and checkers of these codes in CED circuits [20].

2.2. Logical Expressions for the Check Functions of Polynomial Codes

As a rule, the values of check symbols for polynomial codes are obtained using shift registers [18], which is due to the specifics of their application. However, in some applications of polynomial codes, polynomial code encoders representing combinational circuits are required. One example is the design of CED circuits for discrete devices. When constructing an encoder, one needs a formula describing each check symbol of a polynomial code, since this device should be universal and generate a check vector on the incoming data vector in one cycle.

The following trivial method can be used to get formulas for the check symbols of a polynomial code: for the complete set of data vectors with m bits, compute the values of check symbols and form check vectors; then, for each check symbol, obtain logical expressions. Based on such functions, the encoder of the polynomial code under consideration (a combinational circuit) is easily designed.

To get logical expressions for the check functions, it is necessary to determine the values of all check symbols when substituting as arguments the Boolean vectors corresponding to the code data vectors from the complete set 2^m. Here, m denotes the number of arguments. However, this operation becomes rather computationally intensive even for $m \geqslant 6$. The number of data vectors can be reduced to m by utilizing the following result.

Theorem 1. *To obtain functions describing the check symbols of polynomial codes, it is necessary and sufficient to determine the values of check symbols when substituting the arguments from data vectors with weight $r = 1$.*

The proof of this theorem is given in the Appendix.

Theorem 1 leads to an algorithm for determining logical expressions for the check functions of a polynomial code.

Algorithm 1. Rules to form check functions for polynomial codes:

1. Specify a generating polynomial N and the number m of bits in the data vector.

2. Determine the number of bits in the check vectors, $k = \lfloor \log_2 N \rfloor$.

3. Form the set Q of data vectors with weight $r = 1$ and cardinality $|Q| = m$. Rank the data vectors by the significant bit precedence and compile their list.

4. Let $i = 1$.

5. Consider the ith data vector and determine the values of bits of the check vector by division.

6. Compile formulas for check functions: if the value of the check symbol equals 1, then include the ith data bit in the formula; otherwise, do not.

7. Write the bits in formulas through the XOR sign.

8. Let $i := i + 1$.

9. $i + 1 > m$? If "yes," proceed to Step 10; otherwise, get back to Step 5.

10. The formulas have been obtained.

Using Algorithm 1, we obtain the formulas for the check functions of the above polynomial code with $N = 13$ and $m = 6$. All the data vectors with weight $r = 1$ are provided in Table 1.

Table 1. The codewords of a polynomial code with the generating polynomial corresponding to $N = 13$ given $m = 3$, for data vectors with weight $r = 1$

m	y_6	y_5	y_4	y_3	y_2	y_1	g_3	g_2	g_1
6	1	0	0	0	0	0	1	0	1
5	0	1	0	0	0	0	1	1	1
4	0	0	1	0	0	0	1	1	0
3	0	0	0	1	0	0	0	1	1
2	0	0	0	0	1	0	1	0	0
1	0	0	0	0	0	1	0	1	0

Based on Table 1, it is easy to get the formula for each function g_j. One should proceed as follows.

Algorithm 2. Rules to form the check functions of polynomial codes by the codeword table for data vectors with weight $r = 1$:

1. Determine m codewords for the data vectors with weight $r = 1$.

2. Let $j = 1$.

3. Consider the jth column corresponding to the jth check function.

4. Compile the formula by including those data symbols f_i for which 1 is written in the jth column.

5. Let $j := j + 1$.

6. $j+1 > k$? If "yes," proceed to Step 7; otherwise, get back to Step 3.

7. The formulas have been obtained.

We arrive at the following system of Boolean functions:

$$\begin{cases} g_1 = y_3 \oplus y_5 \oplus y_6 \\ g_2 = y_1 \oplus y_3 \oplus y_4 \oplus y_5 \\ g_3 = y_2 \oplus y_4 \oplus y_5 \oplus y_6. \end{cases}$$

Algorithms 1 and 2 yield formulas for any polynomial code with the linear rate of computational intensity. They can be used to design encoders of polynomial codes as part of checkers in CED circuits.

3. A CLASSIFICATION OF POLYNOMIAL CODES BY THE TYPE OF CHECK FUNCTIONS

3.1. Special Properties of Polynomial Codes

The check functions of polynomial codes are linear. The following peculiarities of linear Boolean functions are well known [17].

Theorem 2. *A linear Boolean function is self-dual only if it has an odd number of essential arguments.*

Theorem 3. *A linear Boolean function is self-quasidual only if it has an even number of essential arguments.*

Let us recall several well-known definitions [7].

Definition 1. A function belongs to the class of self-dual Boolean functions if it takes opposite values when inverting all its arguments:

$$SD = \{f(x_1, x_2, \ldots, x_t) | f(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_t}) = \overline{f(x_1, x_2, \ldots, x_t)}\}.$$

Definition 2. A function belongs to the class of self-quasidual[1] Boolean functions if it takes the same values when inverting all its arguments:

$$SQD = \{f(x_1, x_2, \ldots, x_t) | f(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_t}) = f(x_1, x_2, \ldots, x_t)\}.$$

According to Theorems 2 and 3, polynomial codes with different generating polynomials N and with different number m of data symbols belong to one of the three classes: codes whose encoders are described only by self-dual Boolean functions (class I), codes whose encoders are described only by self-quasidual Boolean functions (class II), and codes whose encoders are described by both self-dual and self-quasidual Boolean functions (class III).

Definition 3. Discrete devices whose outputs are described only by self-dual functions are called self-dual devices (SD devices).

Definition 4. Discrete devices whose outputs are described only by self-quasidual functions are called self-quasidual devices (SQD devices).

Definition 5. Discrete devices whose outputs are described, in one part, by self-dual Boolean functions and, in the other part, by self-quasidual Boolean functions are called self-dual/self-quasidual devices (SD/SQD devices).

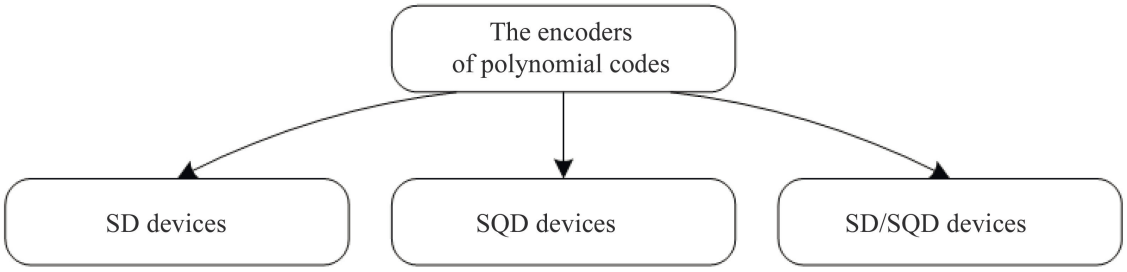

Fig. 2. The encoders of polynomial codes: classification.

Thus, the encoders of polynomial codes may be devices of only three types (Fig. 2).

It is necessary to emphasize the following important peculiarities inherent in the encoders of polynomial codes [17].

Theorem 4. *In the circuit implementation of a linear Boolean function with an odd number of essential arguments, computation control via the self-duality attribute does not check faults of logic gates connected by paths with an even number of inputs.*

Theorem 5. *In the circuit implementation of a linear Boolean function with an even number of essential arguments, computation control via the self-quasiduality attribute does not check faults of logic gates connected by paths with an even number of inputs.*

According to Theorems 4 and 5, it is impossible to design a fully self-checking CED circuit via the attribute of self-duality and/or self-quasiduality of check functions since it is impossible to detect all faults of polynomial encoders when controlling only such attributes. Indeed, the errors caused by faults of XORs in linear circuits are always translated to their outputs, and computation control via the attribute of self-duality or self-quasiduality checks the two-rail property of the spatial signal but not of that coming in parallel to the checker's inputs. Therefore, under the conditions of Theorems 4 and 5, the error is manifested when supplying both sets of argument values to the inputs of the device and is not detected by the checker of a self-dual or self-quasidual signal. Additional control via one more diagnostic attribute is required. The codeword's belonging to the chosen polynomial code may be such an attribute. Indeed, on each set of argument values in the presence of an error, either the correct values of all bits of the check vector will be computed or a distortion will occur at least in one bit. The checker of a polynomial code will detect this event. Thus, polynomial codes can be effectively used in the structure of computation control via several diagnostic attributes, which will be discussed below.

3.2. The Classes of Polynomial Codes by the Encoder Type

Based on the properties of linear functions and the logical expressions for the functions describing the check symbols of polynomial codes (see the simple method above), we have classified polynomial codes by the type of their encoders used in checkers within CED circuits [21]. Table 2 gives a fragment of this classification.

The first column in the table contains the numbers N corresponding to the generating polynomials. Table 2 provides all generating polynomials for constructing polynomial codes with $k = 1, \ldots, 6$ check symbols.

In practice, a convenient representation for different classes of polynomial codes is a matrix form that indicates the belonging of an encoder of a particular code with a given generating polynomial to a certain class under a given number of data symbols. Table 3 presents such a matrix for codes

[1] The author [7] used the somewhat inappropriate term of self-antidual functions to denote self-quasidual ones.

with $k \leqslant 4$ check symbols. Here, the cells at the intersection of particular rows and columns are set off in three different colors: black, gray, and white (meaning that the encoder of the polynomial code belongs to the class of SD devices, SQD devices, and SD/SQD devices, respectively).

Table 2. The detailed classification of polynomial codes by encoder types

N	SD	SQD	SD/SQD
$k = 1$			
3	$m \equiv 1(\mathrm{mod}2)$	$m \equiv 0(\mathrm{mod}2)$	–
$k = 2$			
5	$m \equiv 2(\mathrm{mod}4)$	$m \equiv 0(\mathrm{mod}4)$	$m \equiv 1(\mathrm{mod}2)$
7	$m \equiv 2(\mathrm{mod}3)$	$m \equiv 0(\mathrm{mod}3)$	$m \equiv 1(\mathrm{mod}3)$
$k = 3$			
9	$m \equiv 3(\mathrm{mod}6)$	$m \equiv 0(\mathrm{mod}6)$	$m \equiv \alpha(\mathrm{mod}8)$, $\alpha \in \{0, 1, \ldots, 5\}\backslash\{0, 3\}$
11	$m \equiv 2(\mathrm{mod}7)$	$m \equiv 0(\mathrm{mod}7)$	$m \equiv \alpha(\mathrm{mod}7)$, $\alpha \in \{0, 1, \ldots, 6\}\backslash\{0, 2\}$
13	$m \equiv 6(\mathrm{mod}7)$	$m \equiv 0(\mathrm{mod}7)$	$m \equiv \alpha(\mathrm{mod}7)$, $\alpha \in \{0, 1, \ldots, 6\}\backslash\{0, 6\}$
15	–	$m \equiv 0(\mathrm{mod}4)$	$m \equiv \alpha(\mathrm{mod}7)$, $\alpha \in \{1, 2, 3\}$
$k = 4$			
17	$m \equiv 4(\mathrm{mod}8)$	$m \equiv 0(\mathrm{mod}8)$	$m \equiv \alpha(\mathrm{mod}8)$, $\alpha \in \{0, 1, \ldots, 7\}\backslash\{0, 4\}$
19	$m \equiv 3(\mathrm{mod}15)$	$m \equiv 0(\mathrm{mod}15)$	$m \equiv \alpha(\mathrm{mod}15)$, $\alpha \in \{0, 1, \ldots, 14\}\backslash\{0, 3\}$
21	–	$m \equiv 0(\mathrm{mod}6)$	$m \equiv \alpha(\mathrm{mod}6)$, $\alpha \in \{1, 2, 3, 4, 5\}$
23	$m \equiv 2(\mathrm{mod}14)$	$m \equiv 0(\mathrm{mod}14)$	$m \equiv \alpha(\mathrm{mod}14)$, $\alpha \in \{0, 1, \ldots, 13\}\backslash\{0, 2\}$
25	$m \equiv 8(\mathrm{mod}15)$	$m \equiv 0(\mathrm{mod}15)$	$m \equiv \alpha(\mathrm{mod}15)$, $\alpha \in \{0, 1, \ldots, 14\}\backslash\{0, 8\}$
27	$m \equiv 4(\mathrm{mod}12)$	$m \equiv 0(\mathrm{mod}12)$	$m \equiv \alpha(\mathrm{mod}12)$, $\alpha \in \{0, 1, \ldots, 11\}\backslash\{0, 4\}$
29	$m \equiv 6(\mathrm{mod}14)$	$m \equiv 0(\mathrm{mod}14)$	$m \equiv \alpha(\mathrm{mod}14)$, $\alpha \in \{0, 1, \ldots, 13\}\backslash\{0, 6\}$
31	$m \equiv 4(\mathrm{mod}5)$	$m \equiv 0(\mathrm{mod}5)$	$m \equiv \alpha(\mathrm{mod}5)$, $\alpha \in \{1, 2, 3\}$
$k = 5$			
33	$m \equiv 5(\mathrm{mod}10)$	$m \equiv 0(\mathrm{mod}10)$	$m \equiv \alpha(\mathrm{mod}10)$, $\alpha \in \{0, 1, \ldots, 9\}\backslash\{0, 5\}$
35	$m \equiv 2(\mathrm{mod}42)$	$m \equiv \alpha(\mathrm{mod}42)$, $\alpha \in \{0, 20, 31\}$	$m \equiv \alpha(\mathrm{mod}42)$, $\alpha \in \{0, 1, \ldots, 41\}\backslash\{0, 2, 20, 31\}$
37	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{40, 56\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 38\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 1, \ldots, 61\}\backslash\{0, 38, 40, 56\}$
39	$m \equiv 1(\mathrm{mod}28)$	$m \equiv 0(\mathrm{mod}28)$	$m \equiv \alpha(\mathrm{mod}28)$, $\alpha \in \{0, 1, \ldots, 27\}\backslash\{0, 1\}$
41	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{18, 60\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 39, 62\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 1, \ldots, 61\}\backslash\{0, 18, 39, 60, 62\}$
43	$m \equiv 11(\mathrm{mod}30)$	$m \equiv 0(\mathrm{mod}30)$	$m \equiv \alpha(\mathrm{mod}30)$, $\alpha \in \{0, 1, \ldots, 29\}\backslash\{0, 11\}$
45	–	$m \equiv 0(\mathrm{mod}12)$	$m \equiv \alpha(\mathrm{mod}12)$, $\alpha \in \{1, 2, \ldots, 11\}$
47	$m \equiv 2(\mathrm{mod}31)$	$m \equiv 0(\mathrm{mod}31)$	$m \equiv \alpha(\mathrm{mod}31)$, $\alpha \in \{0, 1, \ldots, 30\}\backslash\{0, 2\}$
49	–	$m \equiv 0(\mathrm{mod}21)$	$m \equiv \alpha(\mathrm{mod}21)$, $\alpha \in \{1, 2, \ldots, 20\}$
51	$m \equiv 5(\mathrm{mod}8)$	$m \equiv 0(\mathrm{mod}8)$	$m \equiv \alpha(\mathrm{mod}8)$, $\alpha \in \{0, 1, \ldots, 7\}\backslash\{0, 5\}$
53	$m \equiv 1(\mathrm{mod}30)$	$m \equiv 0(\mathrm{mod}30)$	$m \equiv \alpha(\mathrm{mod}30)$, $\alpha \in \{0, 1, \ldots, 29\}\backslash\{0, 1\}$
55	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{34, 42\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 10\}$	$m \equiv \alpha(\mathrm{mod}62)$, $\alpha \in \{0, 1, \ldots, 62\}\backslash\{0, 10, 34, 42\}$
57	$m \equiv 17(\mathrm{mod}28)$	$m \equiv 0(\mathrm{mod}28)$	$m \equiv \alpha(\mathrm{mod}28)$, $\alpha \in \{0, 1, \ldots, 27\}\backslash\{0, 17\}$
59	$m \equiv 18(\mathrm{mod}31)$	$m \equiv 0(\mathrm{mod}31)$	$m \equiv \alpha(\mathrm{mod}31)$, $\alpha \in \{0, 1, \ldots, 30\}\backslash\{0, 18\}$
61	$m \equiv 22(\mathrm{mod}31)$	$m \equiv 0(\mathrm{mod}31)$	$m \equiv \alpha(\mathrm{mod}31)$, $\alpha \in \{0, 1, \ldots, 30\}\backslash\{0, 22\}$
63	–	$m \equiv 0(\mathrm{mod}6)$	$m \equiv \alpha(\mathrm{mod}6)$, $\alpha \in \{1, 2, \ldots, 5\}$

Table 3. The matrix of polynomial codes with $k = 1, \ldots, 4$ check symbols

m	N														
	$k=1$	$k=2$		$k=3$				$k=4$							
	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															
27															
28															
29															
30															

4. USING POLYNOMIAL CODES IN THE DESIGN OF SELF-CHECKING DISCRETE DEVICES WITH COMPUTATION CONTROL VIA SEVERAL DIAGNOSTIC ATTRIBUTES

4.1. The Structural Diagram of a CED Circuit

Consider the peculiarities of using polynomial codes in the design of self-checking discrete devices with computation control via several diagnostic attributes. The approach to implementing CED circuits via several diagnostic attributes simultaneously was used earlier in [22] when organizing computation control of devices based on the principle of Boolean signal correction and the properties of balanced codes of the form "r of $2r$," where r specifies the weight of a codeword. It was also investigated in [16] in the context of computation control via several diagnostic attributes using the well-known Hamming codes. Figure 3 shows the structural diagram of computation control via several diagnostic attributes.

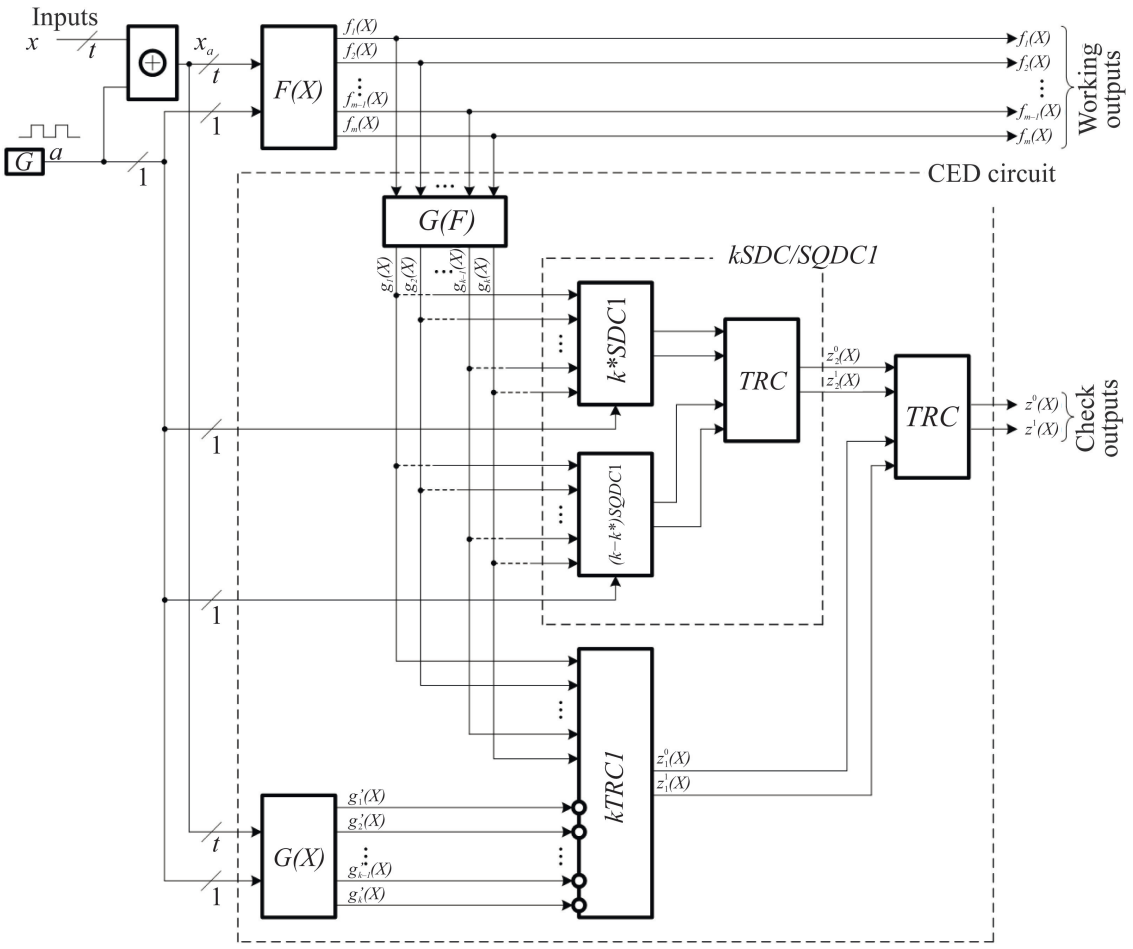

Fig. 3. The structural diagram of a CED circuit for self-dual discrete devices via several diagnostic attributes.

In this structure, the initial discrete device (the object under diagnosis) is a block $F(X)$ intended to compute the values of Boolean functions $f_1(X), f_2(X), \ldots, f_{m-1}(X), f_m(X)$ when supplying the sets of argument values $< X >=< x_t x_{t-1} \ldots x_2 x_1 >$ to the inputs. This object is an SD device. A special CED circuit with several diagnostic attributes is used to control computations at the outputs of the block $F(X)$. The CED circuit is implemented on the basis of polynomial codes. The object's outputs are identified with a data vector with m bits. Given the values of functions $f_1(X), f_2(X), \ldots, f_{m-1}(X), f_m(X)$, the block $G(F)$ forms the values of check symbols $g_1(X), g_2(X), \ldots, g_{k-1}(X), g_k(X)$ for the codewords of a selected polynomial code, thus generating a particular check vector. On the other hand, the block $G(X)$ generates the values of check symbols $g'_1(X), g'_2(X), \ldots, g'_{k-1}(X), g'_k(X)$ for the codewords of the same polynomial code when supplying the sets of argument values $< X >=< x_t x_{t-1} \ldots x_2 x_1 >$ to the inputs. This solution provides a computation control subcircuit based on the belonging of the codewords formed in the CED circuit to the selected polynomial code. More precisely, the same-name outputs of the blocks $G(F)$ and $G(X)$ are connected to the inputs of the two-rail comparator $kTRC1$, which compares k pairs of signals at the inputs and forms a single pair of check signals $z_1^0(X)$ and $z_1^1(X)$. This block is implemented in the two-rail logic and represents a two-rail compression circuit based on $k-1$ elementary *two-rail checkers* (TRC) [23]. Therefore, the signals from one of the blocks $G(F)$ or $G(X)$ are preinverted. (In Fig. 3, these are the signals from $G(X)$ are inverted.) In general, when designing the block $G(X)$, one can immediately receive the inverted signals $g_1(X), g_2(X), \ldots, g_{k-1}(X), g_k(X)$; then inverters at the comparator's inputs will not be needed. Computation errors at the outputs of

the blocks $G(F)$ and $G(X)$, as well as in the comparator's structure, violate the two-rail property of the signal at the outputs $z_1^0(X)$ and $z_1^1(X)$, which indirectly indicates the presence of faults in the structure of the entire system. This structure of computation control (with check symbols supplementing data symbols) has been studied quite well [24].

Since the outputs of the block $G(F)$ are described by self-dual and/or self-quasidual functions, the self-duality/self-quasiduality control subcircuit is used for the signals $g_1(X), g_{k-1}(X), g_k(X)$ to control computations. For self-duality, the block k^*SDC1 is installed to control k^* self-dual signals and generate one pair of two-rail outputs. Similarly, for self-quasiduality, the block $(k-k^*)SQDC1$ is installed to control $k-k^*$ self-quasidual signals and generate one pair of two-rail outputs. These blocks are built based on compression circuits for self-dual and self-quasidual signals [11] and checkers of self-dual and self-quasidual signals [25]. The outputs of both blocks k^*SDC1 and $(k-k^*)SQDC1$ are connected to the inputs of the elementary compression block TRC of two-rail signals. Its outputs are the check outputs $z_2^0(X)$ and $z_2^1(X)$ of the computation control subcircuit via self-duality/self-quasiduality. The outputs of the computation control subcircuits via different diagnostic attributes are connected to the inputs of one block TRC. Its outputs $z^0(X)$ and $z^1(X)$ are the check outputs of the CED circuit.

The operation principles and tuning of control devices for self-dual and self-quasidual signals were described in [25]. The entire structure involves temporal redundancy and the data inversion mode [6]. In this case, signals are represented as pulse sequences: zero is encoded by the sequence $0101\ldots01$ and one by $1010\ldots10$. As a result, the operation of devices is implemented by supplying sets of input argument values in pairs: the first is an operating set, and the second is a check set. The sets of input arguments in a pair are orthogonal in all variables (inverse in all variables). This peculiarity of the computation control approach requires a rectangular pulse generator G, forming a signal a with a pulse ratio of 2. A sequence of signals corresponding to the input variables is obtained using a cascade of two-input XORs: their first inputs receive signals from particular inputs and the second ones the signal a. The same signal can be supplied to the inputs of the blocks $F(X)$ and $G(X)$ when they are designed from non-self-dual structures using the Shannon decomposition with a single variable. The signal a is also required for the operation of self-duality/self-quasiduality checkers.

According to [16], the computation control structure with several diagnostic attributes has better checkability in terms of error observability at the outputs of objects under diagnosis, owing to the operation mode and the possibility of versatile computation control.

Note the following important peculiarity of the computation control structure with several diagnostic attributes.

Theorem 6. *In the CED circuit with computation control via several diagnostic attributes, any single errors are detected at the outputs of the computation control subcircuit by the attributes of the self-duality and/or self-quasiduality of the signals formed.*

The proof of this result is given in the Appendix.

In practice, Theorem 6 simplifies the design of CED circuits with detection of any errors at the outputs of the object under diagnosis by excluding from consideration the object's faults that cause single errors at their outputs.

4.2. An Algorithm for Selecting the Generating Polynomial

The general algorithm for designing a self-checking CED circuit with several diagnostic attributes using polynomial codes contains the following steps.

Algorithm 3. Rules to design a CED circuit with several diagnostic attributes using polynomial codes:

1. Consider the structure of a combinational object under diagnosis with m outputs and t inputs.

2. From the complete set Ω of faults within a given model (e.g., single stuck-at faults at the outputs of logic gates), select a subset $\Omega' \subset \Omega$ of only those occurring at the outputs of logic gates connected by paths with two or more outputs of the object under diagnosis.

3. If $\Omega' = \varnothing$, organize computation control using a polynomial code with a generating polynomial $N = 3$ and design a CED circuit with several diagnostic attributes. If $\Omega' = \varnothing$, consider the possibilities of error detection using polynomial codes with the generating polynomials $N = 3 + 2i$, $i \in \mathbb{N}$. In this case, take only polynomials for which $k = \lfloor \log_2 N \rfloor < m$. In other words, the maximum value of this number is $N = 2^{m-1} - 1$, and the list contains the numbers $N = 3+2i, i = \{1, \ldots, 2^{m-2}-2\}$.

4. Simulate the operation of the self-dual device by supplying all pairs $(A_i^j, B_{2^n-1-i}^j)$ to its inputs, where A_i^j and $B_{2^n-1-i}^j$ are the values of the jth function formed at the jth output of the object under diagnosis under an input combination corresponding to the decimal equivalent of i and $2^n - 1 - i$, sets of argument values, and the impact of single stuck-at faults from the set Ω'. Fix the errors caused by them at the outputs.

5. Compile the list of generating polynomials $N = 3 + 2i$, $i = \{1, 2, \ldots, 2^{m-2} - 2\}$, and arrange them in ascending order of N. With such an ordering, one first considers the generating polynomials yielding codes with the smallest number of check symbols.

6. Take the first member of this list and remove it from there.

7. On the complete family of the sets of argument values, determine the values of the functions defining the check symbols for the codewords of the polynomial code with the chosen generating polynomial N.

8. Check error detection by computation control via the self-duality and/or self-quasiduality attribute of the check functions obtained. From the complete set Ψ of errors occurring at the circuit outputs, eliminate single errors and all errors with multiplicities $d \geqslant 2$, detected by computation control via the self-duality and/or self-quasiduality attribute. Form the set $\Psi' \subset \Psi$ of errors undetectable via this attribute.

9. If $\Psi' = \varnothing$, organize computation control using a polynomial code with the generating polynomial corresponding to the number N. If $\Psi' = \varnothing$, check the detection of errors from the set Ψ' using the polynomial code selected.

10. If all errors are detected, design the CED circuit based on the polynomial code selected; otherwise, take the next generating polynomial from the list (if non-empty) and return to Step 4 of the algorithm. In the case of the empty list, it is impossible to design a fully self-checking structure with computation control via several diagnostic attributes.

Consider an example of computation control using this algorithm for a self-dual discrete device described by the following system of Boolean functions:

$$
\begin{aligned}
f_1 &= \underbrace{x_1x_2}_{1} \vee \underbrace{x_1\overline{x_4}}_{2} \vee \underbrace{x_2\overline{x_4}}_{3},\\
f_2 &= \underbrace{x_3x_4}_{4} \vee \underbrace{x_1x_4}_{5} \vee \underbrace{x_1x_3}_{6},\\
f_3 &= \underbrace{x_1\overline{x}_2\overline{x}_3}_{7} \vee \underbrace{\overline{x_2}x_3x_4}_{8} \vee \underbrace{x_2\overline{x_3}x_4}_{9} \vee \underbrace{x_1x_2x_3}_{10},\\
f_4 &= \underbrace{\overline{x_2}x_3x_4}_{8} \vee \underbrace{x_2\overline{x_3}x_4}_{9} \vee \underbrace{x_2x_3\overline{x_4}}_{11} \vee \underbrace{x_1x_3x_4}_{12} \vee \underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4}_{13};\\
f_5 &= \underbrace{x_1\overline{x_4}}_{2} \vee \underbrace{x_1\overline{x_3}}_{14} \vee \underbrace{x_1\overline{x_2}}_{15} \vee \underbrace{\overline{x}_2\overline{x}_3\overline{x}_4}_{16},\\
f_6 &= \underbrace{x_1\overline{x_3}x_4}_{17} \vee \underbrace{x_1x_3\overline{x_4}}_{18} \vee \underbrace{x_1\overline{x_2}x_3}_{19} \vee \underbrace{\overline{x_1}x_3x_4}_{20} \vee \underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4}_{21}.
\end{aligned}
$$

At Step 1 of the algorithm, we take a particular two-level implementation of the device by the above formulas, with the inversions of arguments transferred to the inputs of the first-cascade gates. Let all non-repeating conjunctions in the expressions be numbered. The structural diagram of this implementation is omitted here due to its bulkiness. It contains 21 multi-input AND gates in the first cascade and 6 multi-input OR gates in the second one.

At Steps 2 and 3, we determine the subset of single stuck-at faults of logic gates causing multiple errors at the device outputs and error detection possibilities using a polynomial code with the generating polynomial $N = 3$. Multiple errors at the device outputs can be caused only by the faults of the gates realizing conjunctions with numbers 2, 8, and 9. (They correspond to possible simultaneous distortions in the values of the functions f_1 and f_5, the functions f_3 and f_4, and the functions f_3 and f_4, respectively.) The polynomial code with the generating polynomial $N = 3$ is not suitable for computation control in the case considered.

At Step 4, we simulate the operation of the device under single stuck-at faults and memorize the results in tabular form.

We compile the list of generating polynomials $N = 3 + 2i,\ i = \{1, \ldots, 14\}$.

Next, we choose the polynomial corresponding to the number $N = 5$, removing it from the list. We check the error detection possibilities at the outputs of the device.

According to the simulation results, the polynomial code with the chosen generating polynomial allows detecting all multiple errors caused by the faults of the gates realizing conjunctions with numbers 2, 8, and 9, except for five double errors caused by a stuck-at-1 fault of the gate realizing conjunction with number 2. (These five faults are detected neither by self-duality control of the functions describing the check symbols of the codewords (see the matrix in Table 3) nor by the control by the belonging of the check vector to this polynomial code.) The cardinality of the set is $|\Psi'| \neq 5$. Hence, it is necessary to choose the next polynomial from the updated list. This polynomial corresponds to the number $N = 7$.

We perform the same actions as before. Here, judging by the matrix in Table 3, computations are controlled via the following attributes: the belonging of the functions describing the check symbols of the codewords to the class of self-quasidual ones and the belonging of the check vector to the codewords of the polynomial code selected. According to the simulation results, all multiple errors are detected by computation control via only one of the diagnostic attributes. Next, we design the CED circuit with two diagnostic attributes using the polynomial code selected.

Note that computation control via the attributes of self-duality and self-quasiduality insignificantly complicates CED circuits compared to CED circuits with computation control via polynomial codes: the compression schemes of self-dual/self-quasidual signals, as well as the checkers of these signals, have a rather simple structure [11, 25]. Therefore, it is possible to design less redundant structures than, e.g., in the case of duplication, while obtaining improved checkability in terms of error observability at the outputs of CED circuits.

An advantage of Algorithm 3 is that, with an appropriate choice of the generating polynomial from small $N = 3$ to $N = 2^{m-2} - 2$, it is possible to achieve the full detection of errors using codes with small k, which also affects the performance of CED circuit implementation. A drawback is the necessity of fault modeling and error detection at the outputs of objects under diagnosis, which limits the applicability of this method for a sufficiently large number of inputs. The applicability of the method for CED circuit design in some discrete devices also depends on the complexity of Boolean functions computed at their outputs [26]. The complexity of the algorithm for constructing a self-checking device can be asymptotically estimated as $2^{O(t)}$, i.e., the problem is solved in exponential time with a linear index. Nevertheless, it is always possible to construct CED circuits using decomposition methods. Besides, considering the structures of objects under diagnosis is

an effective CED circuit design method when standard modular redundancy-based ones are not applied.

Note also that polynomial codes detect a large number of errors under high degrees of the polynomials $M(y)$ and $G(y)$, which is widely used in computation control and data transmission. However, in the above application of polynomial codes, the degree of the polynomial $M(y)$ is determined by the number of outputs in the object under diagnosis, whereas the degrees of the generating polynomials are selected considering the detection of errors caused by faults in the structures of such objects. As a matter of fact, the earlier the number N is found by Algorithm 3, the smaller the degree of the residue $R(y)$ will be, and the less structural redundancy the final self-checking discrete device will have.

5. CONCLUSIONS

Compared to the widely known approaches, the design method proposed above endows the resulting fully self-checking discrete devices with improved checkability indicators. Moreover, the method allows detecting a subset of errors in computations under fixed input actions. This is due to the binary-sequence representation of signals in discrete devices.

The CED circuit design algorithm developed above based on the properties of polynomial codes involves a sequential search of generating polynomials from the smallest, associated with the number $N = 3$, to the corresponding number $N = 2^{m-2} - 2$. Under $m < 6$, the number of generating polynomials satisfying this condition is small, reducing the possibilities for constructing a fully self-checking device. Hence, the approach under consideration should be most effective for devices with sufficiently many outputs: $m \geqslant 6$. As N grows to the limit, the number of check symbols in the codewords changes from $k = 1$ to $k = m - 1$, also increasing the cardinality of the set of errors detected by polynomial codes. Therefore, the algorithm will be effective when constructing self-checking devices for almost any structures of objects under diagnosis with $m \geqslant 6$. Nevertheless, for each particular discrete device, it is necessary to compare the indicators of their self-checking implementation with those of self-checking implementations obtained by other methods, e.g., duplication.

Possible directions of further research include setting various experiments with test discrete devices and estimating the efficiency indicators of their self-checking implementations, developing other algorithms for choosing generating polynomials considering the structural peculiarities of objects under diagnosis and the peculiarities of other linear codes in CED circuit design via several diagnostic attributes, and studying the peculiarities of constructing such devices on different (particularly programmable) element bases.

As we believe, the use of computation control via several diagnostic attributes in the design of self-checking discrete control devices is a promising and underinvestigated approach to the realization of components of highly reliable control systems.

APPENDIX

Proof of Theorem 1.

Necessity. Each check function is described by the general expression $g_j = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus f_{i_q}$, $j \in \{1, k\}$, $i_1, i_2, \ldots, i_q \in \{1, \ldots, m\}$. To obtain detailed expressions for each check function, it is necessary to establish significant data symbols. Consider all data vectors with weight $r = 1$ to determine sequentially the values of the check symbols by substituting a single significant data symbol. If the function value is $g_j = 0$, the corresponding data symbol will not affect the particular check function; otherwise ($g_j = 1$), it will be included in the logical expression describing the check function.

Sufficiency. When substituting the data vector with weight $r = 1$, the value of the function g_j will be determined only by the value of the significant data symbol; hence, it will be included in the descriptive formula for the corresponding check function. Consider all data vectors with weight $r = 1$ to determine all arguments summed in each function g_j. This yields the formulas describing the check symbols of polynomial codes.

Proof of Theorem 6.

All functions of the encoder of a polynomial code are linear: $g_j = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus f_{i_q}$, $j \in \{1, k\}$, $i_1, i_2, \ldots, i_q \in \{1, \ldots, m\}$. Consider one of them and suppose the occurrence of a single error at the output $f_q \in \{1, \ldots, m\}$. Then the value of the function f_p is distorted, becoming equal to $f_p = 0$ or $f_p = 1$. The original expression for the check function is $g_j = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus f_p \oplus \ldots \oplus f_{i_q}$. Due to the distortion, it takes the form $g_j = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus (f_p = 0) \oplus \ldots \oplus f_{i_q} = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus f_{i_q}$ or $g_j = f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus (f_p = 1) \oplus \ldots \oplus f_{i_q} = \overline{f_{i_1} \oplus f_{i_2} \oplus \ldots \oplus f_{i_q}}$. In the first case, we have a parity change in the number of arguments of the original function and its dependence on all variables except f_p. In the second case, the parity of the number of arguments changes as well but, besides, the function of mod 2 addition is inverted, which preserves all the attributes of belonging to the classes of self-dual and/or self-quasidual functions. In other words, the function has turned from self-quasidual/self-dual into self-dual/self-quasidual. Due to the new type, the function will be detected in the computation control subcircuit with the attribute selected: it will change under a single error.

REFERENCES

1. Bennetts, R.G., *Design of Testable Logic Circuits*, Addison-Wesley, 1984.
2. Sogomonyan, E.S. and Slabakov, E.V., *Samoproveryaemye ustroistva i otkazoustoichivye sistemy* (Self-Checking Devices and Fault-Tolerant Systems), Moscow: Radio i Svyaz', 1989.
3. Drozd, A., Kharchenko, V., Antoshchuk, S., et. al., Checkability of the Digital Components in Safety-Critical Systems: Problems and Solutions, *Proceedings of the 9th IEEE East-West Design and Test Symposium (EWDTS'2011)*, Sevastopol, Ukraine, 2011, pp. 411–416. https://doi.org/10.1109/EWDTS.2011.6116606
4. Drozd, A.V., Kharchenko, V.S., Antoshchuk, S.G., et al., *Rabochee diagnostirovanie bezopasnykh informatsionno-upravlyayushchikh sistem* (Operational Diagnosis of Safe Information and Control Systems), Khar'kov: Zhukovskii National Aerospace University, 2012.
5. Sapozhnikov, V.V. and Sapozhnikov, Vl.V., *Samoproveryaemye diskretnye ustroistva* (Self-Checking Discrete Devices), St. Petersburg: Energoatomizdat, 1992.
6. Aksenova, G.P., Restoration in Duplicated Units by the Method of Data Inversion, *Avtomat. Telemekh.*, 1987, no. 10, pp. 144–153.
7. Shalyto, A.A., Modules Universal in the Class of Self-dual Functions and in Close Classes, *Izv. Akad. Nauk. Teor. Sist. Upravlen.*, 2001, no. 5, pp. 110–120.
8. Saposhnikov, Vl.V., Dmitriev, A., Goessel, M., and Saposhnikov, V.V., Self-Dual Parity Checking—a New Method for On-line Testing, *Proceedings of the 14th IEEE VLSI Test Symposium*, USA, Princeton, 1996, pp. 162–168. https://doi.org/10.1109/VTEST.1996.510852
9. Gessel', M., Moshanin, V.I., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., Fault Detection in Self-Test Combination Circuits Using the Properties of Self-Dual Functions, *Avtomat. Telemekh.*, 1997, no. 12, pp. 193–200.
10. Gessel', M., Dmitriev, A.V., Sapozhnikov, V.V., and Sapozhnikov, Vl.V., A Functional Fault-Detection Self-test for Combinational Circuits, *Autom. Remote Control*, 1999, vol. 60, no. 11, pp. 1653–1663.
11. Dmitriev, A., Saposhnikov, V., Saposhnikov, V., and Goessel, M., New Self-Dual Circuits for Error Detection and Testing, *VLSI Design*, 2000, vol. 11, no. 1, pp. 1–21. https://doi.org/10.1155/2000/84720

12. Sapozhnikov, V.V., Sapozhnikov, Vl.V., and Gessel', M., *Samodvoistvennye diskretnye ustroistva* (Self-Dual Discrete Devices), St. Petersburg: Energoatomizdat, 2001.

13. Sapozhnikov, V.V., Sapozhnikov, Vl.V., and Valiev, R.Sh., *Sintez samodvoistvennykh diskretnykh sistem* (Design of Self-Dual Discrete Systems), St. Petersburg: Elmor, 2006.

14. Goessel, M., Ocheretny, V., Sogomonyan, E., and Marienfeld, D., *New Methods of Concurrent Checking*, 1st ed., Dordrecht: Springer, 2008.

15. Yablonskii, S.V., *Vvedenie v diskretnuyu matematiku* (Introduction to Discrete Mathematics), Sadovnichii, V.A., Ed., Moscow: Vysshaya Shkola, 2003.

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. https://doi.org/10.15622/ia.22.2.5

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. https://doi.org/10.17223/19988605/65/13

18. Sellers, F.F., Hsiao, M.-Y., and Bearnson, L.W., *Error Detecting Logic for Digital Computers*, New York: McGraw-Hill, 1968.

19. Efanov, D., Plotnikov, D., Sapozhnikov, V., Sapozhnikov, Vl., and Abdullaev, R., Experimental Studies of Polynomial Codes in Concurrent Error Detection Systems of Combinational Logical Circuits, *Proceedings of the 16th IEEE East-West Design and Test Symposium (EWDTS'2018)*, Kazan, Russia, September 14–17, 2018, pp. 184–190. https://doi.org/10.1109/EWDTS.2018.8524684

20. Abdullaev, R. and Efanov, D., Polynomial Codes Properties Application in Concurrent Error-Detection Systems of Combinational Logic Devices, *Proceedings of the 19th IEEE East-West Design and Test Symposium (EWDTS'2021)*, Batumi, Georgia, September 10–13, 2021, pp. 40–46. https://doi.org/10.1109/EWDTS52692.2021.9580992

21. Efanov, D.V. and Pivovarov, D.V., Organization of Control of Calculations at the Outputs of Self-Dual Digital Devices Using Cyclic Redundant Codes, *Trudy 14-go Vserossiiskogo soveshchaniya po problemam upravleniya* (Proceedings of the 14th All-Russian Meeting on Control Problems), Novikov, D.A., Ed., Moscow, June 17–20, 2024, pp. 2395–2399.

22. Efanov, D.V. and Pivovarov, D.V., The Hybrid Structure of a Self-Dual Built-In Control Circuit for Combinational Devices with Pre-Compression of Signals and Checking of Calculations by Two Diagnostic Parameters, *Proceedings of the 19th IEEE East-West Design and Test Symposium (EWDTS'2021)*, Batumi, Georgia, September 10–13, 2021, pp. 200–206. https://doi.org/10.1109/EWDTS52692.2021.9581019

23. Lala, P.K., *Self-Checking and Fault-Tolerant Digital Design*, San Francisco: Morgan Kaufmann, 2001.

24. Sahana, A.R., Chiraag, V., Suresh, G., Thejaswini, P., and Nandi, S., Application of Error Detection and Correction Techniques to Self-Checking VLSI Systems: An Overview, *Proceedings of 2023 IEEE Guwahati Subsection Conference (GCON)*, Guwahati, 2023. https://doi.org/10.1109/GCON58516.2023.10183449

25. Efanov, D.V. and Pivovarov, D.V., Checkers of Self-Dual and "Close in Meaning" Signals, *Journal of Instrument Engineering*, 2024, vol. 67, no. 1, pp. 5–19. https://doi.org/10.17586/0021-3454-2024-67-1-5-19

26. Korshunov, A.D., Computational Complexity of Boolean Functions, *Russian Math. Surveys*, 2012, vol. 67, no. 1, pp. 93–165. https://doi.org/10.1070/RM2012v067n01ABEH004777

This paper was recommended for publication by L.Yu. Filimonyuk, a member of the Editorial Board