On the self-testing (m,n)-code checker design

N Butorina[1], Yu Burkatovskaya[2] and E Pakhomova[1]
[1] Institute of Applied Mathematics and Computer Science, National Research Tomsk State University, Tomsk, Russia
[2] School of Computer Science and Robotics, National Research Tomsk Polytechnic University, Tomsk, Russia
E-mail: tracey@tpu.ru

Abstract. We propose an approach to a self-testing (m, n)-code checker design, based on subdividing the set of all code words into special subsets called segments. The checker circuit is constructed by using one- and two-output configurable logic blocks (CLB). Previously, in each output of a CLB, a function representing exactly one segment was implemented. In the proposed approach, at each CLBs output, it is possible to implement functions that represent several segments and to provide the self-testing property. It allows reducing the number of CLBs and simplifying the circuit of the checker.

1. Introduction

In self-checking circuits, code checkers are used to provide the self-testing property. As a rule, the checker is designed on the same element base as the circuit. In the checker, the weight of an input codeword is usually calculated. For this purpose, threshold elements or parallel counters are used [1–6]. This paper proposes a design method of the checker based on configurable logic blocks (CLBs). In papers [7, 8], the authors propose a method for design a self-checking finite-state machine. In the case of correct operation of the circuit, the code words of some unordered code are implemented at the output of the circuit. Usually, the (*m*,*n*)-code is used, where n is the length of a code word, m is the number of unity components (the weight). The papers also develop a method for self-testing (*m*,*n*)-code checker design based on the repeated use of a special decomposition formula for a set of the (*m*,*n*)-code words and implementing the obtained formula by CLBs. In the method of self-testing checker design proposed in [8], the presence of two-output CLBs is not necessary. This means that the method allows building self-testing checkers using any modern CLBs (Field Programmable Gate Arrays (FPGA), manufacturers Xilinx, Altera, Achronix, Actel, Atmel, Lattice semiconductor, etc.).

There is a special requirement for the implementation of the formula: the checker of the (*m*,*n*)-codes must be self-testing for a given set of faults, say V. The set V includes all multiple stuck-at faults occurred at CLBs inputs and outputs. In this case, only one CLB in the checker can be defective. It is assumed that in a system consisting of a self-testing circuit and a (*m*,*n*)-code checker, either the circuit or the checker can be faulty, but not both.

The self-testing checker should satisfy the following requirements:

1) when a non-code word appears at the circuit output (or at the checker input), the checker should issue a corresponding signal;

2) in the checker itself, a fault from the considered set of faults V may occur, which must be detectable in the working area of the detector, i.e. on the set of all its code words. This means that there must be an (*m*,*n*)-code word for which this fault appears at the outputs of the checker.

The self-testing checker has two outputs with the following combinations of signal values:
a) (01) or (10) mean that the input word is a word of the (*m*,*n*)-code and the checker is faultless;
b) (00) or (11) mean that either the input word is non-code or the checker is faulty.
The checker is designed according to a formula representing the set of all (*m*,*n*)-code words, being the implementation of this formula.

2. The decomposition method for the set of (m,n)-code words

The number of all (m,n)-code words is equal to C_n^m, i.e., the number of combinations of n elements taken m at a time. The code words can be represented by a disjunction of conjunctions of the rank n. We denote this disjunctive normal form (DNF) as $D_n^m(X)$, where X = {x1, x2, …, xn} is the set of variables. If, for example, n = 10, m = 5, then $D_{10}^5(X)$ includes $C_{10}^5 = 10!/(5!\cdot 5!) = 252$ conjunctions of the rank 10 and 2520 symbols. The DNF can not be reduced: $D_n^m(X)$ is both the perfect DNF and the sum of prime implicants, since any two conjunctions of $D_n^m(X)$ are orthogonal by at least two variables.

To represent all (*m*, *n*)-code words, a special decomposition formula was proposed in [9]. We divide the set of variables *X* into two subsets X^1, X^2, where $X^1 = \{x_1, ..., x_g\}$, $X^2 = \{x_{g+1}, ..., x_n\}$. The whole set of the (*m*,*n*)-code words of can be represented by the formula

$$D_n^m(X) = \sum_{i=0}^{m} D_g^i(X^1) D_{n-g}^{m-i}(X^2), \quad (1)$$

where the $\wedge$ between $D_g^i(X^1)$ and $D_{n-g}^{m-i}(X^2)$ is omitted.

Here $D_g^i(X^1)$, $D_{n-g}^{m-i}(X^2)$ are the decomposition functions; we call the cardinality of variable subsets after the last use of formula (1) as the decomposition base and denote it as *k*.

In paper [8], we propose to choose *g* as the least integer number which is greater than or equal to $n/2$, i.e., $g = \lceil n/2 \rceil$. If $g > k$ and/or $n - g > k$, then formula (1) is used again for any decomposition function $D_g^i(X^1)$, $D_{n-g}^{m-i}(X^2)$, $i = \overline{0,m}$ etc. Upon doing that, we obtain the formula for all (*m*,*n*)-code words, where $p \le k$ for all $D_p^q(X^r)$. Multiple decomposition is applied both for the first and to the second multipliers in formula (1).

In this paper, we use only one decomposition step. For example, for D_{14}^7 and $k = 7$, where k is the number of CLBs inputs, we have two subsets of variables

$$X^1 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}, \quad X^2 = \{x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}\}.$$

Formula (1) has the form (2):

$$\begin{aligned} D_{14}^7 = {} & D_7^0(X^1) D_7^7(X^2) \vee D_7^1(X^1) D_7^6(X^2) \vee D_7^2(X^1) D_7^5(X^2) \vee D_7^3(X^1) D_7^4(X^2) \vee \\ & \vee D_7^4(X^1) D_7^3(X^2) \vee D_7^5(X^1) D_7^2(X^2) \vee D_7^6(X^1) D_7^1(X^2) \vee D_7^7(X^1) D_7^0(X^2). \end{aligned} \quad (2)$$

To ensure the self-testing property of the checker, special requirements will be used for the functions implemented by the CLBs. Without loss of generality, we will consider single-output CLBs with not more than 7 inputs and two-output CLBs with not more than 6 inputs.

To provide the self-testing property in the considered class of faults, we use special functions.

Let us denote the set of Boolean vectors where the function $f(x_1, \ldots, x_n)$ is equal to 1 as $S_1(f)$. Let us represent the Boolean function by the table where the columns correspond to Boolean variables, and the rows present Boolean vectors from the set $S_1(f)$.

Definition 1. We call a function *f* as *function of type 2,* if in the table representation, every column contains exactly one unity component, and the number of unity components in all rows is the same.

Table 1 demonstrates an example of a type 2 function.

Theorem 1. For a CLBs output, implementing a type 2 function, and for any input multiple-stuck-at fault, either a test from $S_1(f)$ exists, or the multiple fault manifests itself as an output stuck-at-1 fault at the CLB.

Theorem 2. If a type 2 function is implemented on one of the outputs of a two-output CLB, then whatever the second function associated with the CLB, an input multiple-stuck-at fault of the CLB is either detected on the corresponding output type 2 function when the CLB receives a vector from $S_1(f)$ in the input, or the fault manifests itself as a stuck-at-1 fault.

Theorems 1 and 2 are proven in [9]

Table 1. Example of a type 2 function.

x_1	x_2	x_3	x_4	x_5	x_6
1	1	0	0	0	0
0	0	1	1	0	0
0	0	0	0	1	1

3. Identifying the properties of configurable logic blocks implementing the (m,n)-code words

Definition 2. We call as a *segment* $F_k^q(x_1,...,x_k)$ the set of all (q,k)-code words, i.e., all Boolean vectors where the function $D_k^q(x_1,...,x_k)=1$.

We denote as *FrSet*(x_1, …, x_k) the set of all segments $F_k^q(x_1,...,x_k)$, i.e., the set *FrSet*(x_1, …, x_k) includes all Boolean vectors of the length k and the weight q, where $0 \le q \le k$.

We denote as *FrSubset*(x_1, …, x_k) a subset of the set *FrSet*(x_1, …, x_k) satisfying the following conditions:

1) It contains at least two segments;

2) if we arrange the weights of the segments in ascending order then in the sequence of weights there will be two neighboring elements with the difference at least two.

Examples. A union of segments with the weights 2, 4, 5 is a *FrSubset*(x_1, …, x_k), a union of segments with the weights 2, 3, 4 is not a *FrSubset*(x_1, …, x_k).

Definition 3. Let us call a function whose domain of unity values coincides with a subset *FrSubset*(x_1, …, x_k), *as Fsubset*(x_1, …, x_k).

Table 2 demonstrates an example of such a function.

Table 2. Example of a function Fsubset(x_1,x_2,x_3).

x_1	x_2	x_3
0	0	0
0	1	1
1	0	1
1	1	0

The subset *FrSubset*(x_1,x_2,x_3) for the function from table°2 consists of two segments: $F_3^0(x_1,x_2,x_3)$, $F_3^2(x_1,x_2,x_3)$.

Subsets *FrSubset*(x_1, …, x_p) can be of different cardinality.

Let a CLB implement a function *Fsubset*(x_1, …, x_k). Here k is the number of CLBs inputs.

Further we represent an input multiple stuck-at fault by a *ternary vector* with the components from the set {0, 1, – }. Components equal to 0 or 1 are called as *outer components*, they correspond to inputs with stuck-at-0 or stuck-at-1 faults. Components equal to – are called *inner components*, they correspond to faultless inputs.

Example. Let a CLB have 6 inputs. If it has stuck-at-0 fault at the inputs {x_1,x_4} and stuck-at-1 fault at the input {x_3} then this multiple fault is described by the vector $0-10--$.

Theorem 3. If a CLB implements a function *Fsubset*(x_1, ..., x_k), where k is the number of the CLBs inputs, then for a multiple input stuck-at fault, either there is a test from the set *FrSubset*(x_1, ..., x_k), or a multiple fault manifests itself as a stuck-at-1 fault at the CLBs output.

Proof. Let the CLB implement the function *Fsubset*(x_1, ..., x_k), k be the number of the CLBs inputs. Let the fault be described by a ternary vector β and the vector has r outer components, $r \leq k$. Consider all possible cases.

1. Let $r = k$, i.e., all the components of β are outer. So,

   а) if β is a code word from *FrSubset*(x_1, …, x_k) then the fault manifests itself as the output stuck-at-1 fault;

   б) otherwise, if β is not a code word from *FrSubset*(x_1, …, x_k), then the fault is detected by any vector from *FrSubset*(x_1, …, x_k); the output value is equal to 0 instead of 1.

2. Let $r < k$, s be the number of unity components of the vector β. There can be several cases:

   а) if $0 < s \leq r < k$, then any vector α from *FrSubset*(x_1, …, x_k) having at least one zero component with the same number as a unity component of β, is a test for the fault. Such vector α exists, because for any component, there is a vector from *FrSubset*(x_1, …, x_k) in which the component is equal to zero;

   б) if $s = 0$, then any vector α from *FrSubset*(x_1, …, x_k), having at least one unity component with the same number as a zero component of β, is a test for the fault. Such vector α exists, because for any component, there is a vector from *FrSubset*(x_1, …, x_k) in which the component is equal to unity.

The theorem is proved.

Theorem 4. If at one of the outputs of two-output CLB a function of the type *Fsubset*(x_1, ..., x_k) is implemented, whatever the second function associated with the CLB, a multiple input stuck-at fault of this CLB is either detected on the corresponding vector α from *FrSubset*(x_1, ..., x_k), or it appears on this output as a stuck-at-1 fault.

Proof. At one of the CLBs outputs, a function of the type *Fsubset*(x_1, ..., x_k) is implemented. According to Theorem 1, for a multiple input stuck-at fault , either there is a test from the set *FrSubset*(x_1, ..., x_k), or the multiple fault manifests itself as an output stuck-at-1 fault. Both functions implemented by the CLB depend on the same sets of variables , and the functions themselves are implemented on separate memory blocks (LUT). At the second output of the CLB, the multiple stuck-at fault may not appear at all. It depends on the type of function implemented by the second output. The theorem is proved.

Consider the subcircuit 1 (Figure 1). The lower level of this subcircuit consists of a one- or two-output CLB (CLB1), which implements the function (one of the outputs) *Fsubset*(x_1, ..., x_k), and the output of this CLB corresponds to the input of several CLBs implementing the function of type 2 (CLB2 and CLB3). The set of CLBs variables that implement the function of type 2, are different. The outputs of the subcircuit are the outputs of the CLB, implementing a function of type 2.

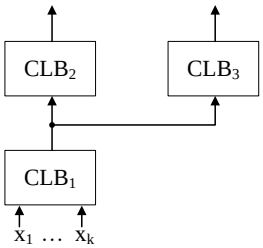

Figure 1. Subcircuit 1.

Theorem 5. For a multiple input stuck-at fault of one of the CLB of subcircuit 1, either there is a test from the set *FrSubset*(x_1, ..., x_k), or the fault manifests itself as an output stuck-at-1 fault of subcircuit 1.

The statement follows from Theorems 1 – 4.

The detector circuit has a tree structure, on the lower level of which only functions $D_p^q(x_1, \ldots, x_p)$ are used, where $0 < q \le p$, $p \le k$, k – is the number of the CLBs inputs. Earlier, in [7, 8], at every CLBs output exactly one function $D_p^q(x_1, \ldots, x_p)$ was implemented. Theorem 5 allows simplifying the self-testing-checker circuit and to reduce the number of CLBs in it. For example, Figure 2 demonstrates "old" checker D_{14}^7, as Figure 3 shows a "new" circuit of the same checker D_{14}^7.

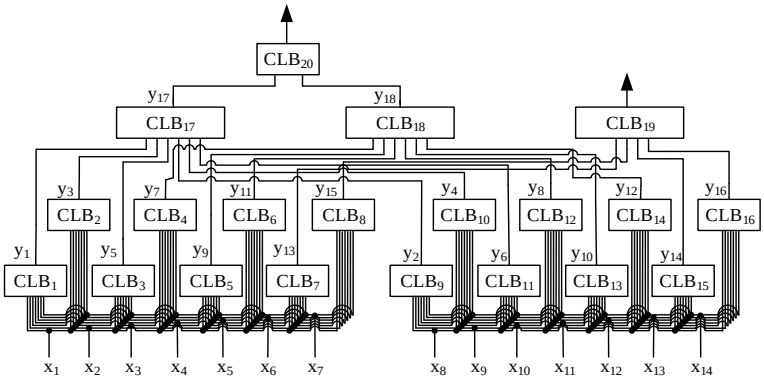

Figure 2. The checker of the (7, 14)-code.

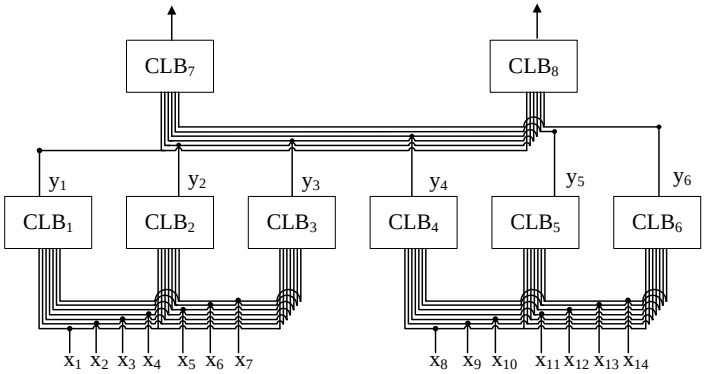

Figure 3. The new checker of the (7, 14)-code.

Consider the "old" circuit. Let us construct a checker for D_{14}^7, using subcurcuit 2 if $k = 7$. At the first step of decomposition one can obtain.

$$D_{14}^7 = D_7^0(X^1)D_7^7(X^2) \vee D_7^1(X^1)D_7^6(X^2) \vee D_7^2(X^1)D_7^5(X^2) \vee D_7^3(X^1)D_7^4(X^2) \vee \\ \vee D_7^4(X^1)D_7^3(X^2) \vee D_7^5(X^1)D_7^2(X^2) \vee D_7^6(X^1)D_7^1(X^2) \vee D_7^7(X^1)D_7^0(X^2). \quad (2)$$

There are 8 Boolean functions depending on the variables x_1, x_2, x_3, x_4, x_5, x_6, x_7. For their implementation one needs 8 CLBs. Their characteristics are listed in table 3.

Table 3. Boolean functions depending on the variables x_1, x_2, x_3, x_4, x_5, x_6, x_7.

№	Implemented function	Output	№	Implemented function	Output
1	$D_7^0(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_1	5	$D_7^4(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_9
2	$D_7^1(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_3	6	$D_7^5(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_{11}
3	$D_7^2(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_5	7	$D_7^6(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_{13}
4	$D_7^3(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_7	8	$D_7^7(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$	y_{15}

There are 8 Boolean functions depending on the variables x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}. For their implementation one needs 8 CLBs. Their characteristics are listed in table 4.

Table 4. Boolean functions depending on the variables x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}.

№	Implemented function	Output	№	Implemented function	Output
9	$D_7^7(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_2	13	$D_7^3(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_{10}
10	$D_7^6(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_4	14	$D_7^2(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_{12}
11	$D_7^5(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_6	15	$D_7^1(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_{14}
12	$D_7^4(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_8	16	$D_7^0(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_{16}

To accomplish formula (2) one needs 4 CLBs. Their characteristics are listed in table 5.

Table 5. Boolean functions for accomplishing formula (2).

№	Implemented function	Output
17	$y_1y_2 \vee y_3y_4 \vee y_5y_6$	y_{17}
18	$y_7y_8 \vee y_9y_{10} \vee y_{11}y_{12}$	y_{18}
19	$y_{13}y_{14} \vee y_{15}y_{16}$	Outputs of the checker
20	$D_2^1(y_{17},y_{18})$	Outputs of the checker

The outputs of CLB_{19} and CLB_{20} are the outputs of the checker. Hence, 20 CLBs are used to implement the checker. The circuit is represented in Figure 2.

At the “new” circuit, we use the following functions (tables 6 and 7).

Table 6. Boolean functions depending on the variables x_1, x_2, x_3, x_4, x_5, x_6, x_7.

№	Implemented function	Output
1	$D_7^0(x_1,x_2,...,x_7)$, $D_7^3(x_1,x_2,...,x_7)$, $D_7^6(x_1,x_2,...,x_7)$	y_1
2	$D_7^1(x_1,x_2,...,x_7)$, $D_7^4(x_1,x_2,...,x_7)$, $D_7^7(x_1,x_2,...,x_7)$	y_3
3	$D_7^2(x_1,x_2,...,x_7)$, $D_7^5(x_1,x_2,...,x_7)$	y_5

Table 7. Boolean functions depending on the variables x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}.

№	Performing function	Output
4	$D_7^7(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$, $D_7^2(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$, $D_7^1(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_2
5	$D_7^6(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$, $D_7^4(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$ $D_7^0(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_4
6	$D_7^3(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$, $D_7^5(x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14})$	y_6

To accomplish formula (2) one needs 2 CLBs. Their characteristics are listed in table 8.

Table 8. Boolean functions for accomplishing formula (2).

№	Performing function	Output
7	$y_1y_2 \vee y_3y_4 \vee y_5y_6$	outputs of the checker
8	$y_1y_4 \vee y_3y_6 \vee y_5y_2$	outputs of the checker

The outputs of CLB_7 and CLB_8 are the outputs of the checker. Hence, to perform checker 8 CLBs are used. The circuit of the detector is represented in Figure 3.

4. Conclusion

Thus, we propose an approach to design of a self-testing (*m*,*n*)-code checker, which allows reducing the number of CLBs and simplifying the checker circuit. Previously, at each output of the CLB, functions representing exactly one segment. In the proposed approach, at each CLBs output, it is possible to implement functions that represent several segments.

References

[1] Parag K. Lala 2000 *Self-Checking and Fault-Tolerant Digital Design* (San Francisco: Morgan Kaufmann) p 400
[2] Anderson D A and Metze G 1973 *IEEE Trans. Computers* C **22** 263–269
[3] Marouf M A and Friedman A D 1978 *IEEE Trans. Computers* C **27** 482–490
[4] Ubar R, Raik J and Vierhaus H-T 2011 *Design and Test Technology for Dependable Systems-on-Chip* (New York: IGI Global) p 578
[5] Göessel M, Ocheretny V, Sogomonyan E and Marienfeld D 2008 *New Methods of Concurrent Checking, Edition 1* (Dordrecht: Springer Science+Business Media B. V.) p 182
[6] Sapozhnikov V V, Sapozhnikov V V, Efanov D V and Pivovarov D V 2017 *Radioelectronics and Computer Science (Radioelektronika i informatika)* **3** 15–22
[7] Matrosova A Yu and Nikitin K V 2003 *Tomsk State University Journal, Supplement* **6** 124–136
[8] Butorina N B and Tsidendorzhieva S R 2008 *Proceeding of 7°Russian conference with international participation "New information technologies in the study of complex structures* (Tomsk: TSU) p 44
[9] Burkatovskaya Yu B, Butorina N B and Matrosova A Yu 2006 *Tomsk State University Journal, Supplement* **17** 190–197