Automatic Control and Computer Sciences
Vol.37, No.2, pp.40-45, 2003

Avtomatika i Vychislitel'naya Tekhnika
Vol.37, No.2, pp.48-55, 2003

SYNTHESIS OF SELF-CHECKING CONTROLLERS BASED ON MODIFIED (m, n)-CODE

V.I. OSTROVSKII, *Assistant Professor, Cand.Sc. (Technical Sciences)*
I.S. LEVIN, *Professor, Cand.Sc. (Technical Sciences)*
S.A. OSTANIN, *Cand.Sc. (Technical Sciences)*
Tel Aviv University
Ramat Aviv 69978 Tel Aviv (Israel)
e-mail: ilial@post.tau.ac.il, serg_os@yahoo.com

The present article proposes a method of constructing self-checking controllers based on a novel code that makes it possible to detect any unidirectional errors in a check word. Unlike traditional codes, the proposed code makes possible a reduction in hardware costs due to an increase in the number of check bits. In test examples this method makes possible a roughly 50% decrease in hardware costs from that in traditional solutions while maintaining the properties of self-checkability.

1. INTRODUCTION

Use of self-checking circuits [1], that is, circuits capable of detecting faults in the course of operation is one of the more effective techniques of increasing the reliability of discrete devices. A broad class of faults in discrete devices produces unidirectional errors at their outputs [2]. By a unidirectional error is understood a distortion (i.e., change in values to their opposite) of an arbitrary number of either only zero or only unit bits of the word.

Most methods of constructing self-checking circuits are based on the use of corresponding codes that detect errors. The entire set of codes may be provisionally divided into two classes, universal codes and context-oriented codes. In the construction of universal codes no constraints are placed on the set of check words, i.e., it is assumed that the set may contain all 2^k words, where k is the number of bits in the check words. In context-oriented codes, it is assumed that some of the 2^k words are not used and, as a result, it becomes possible to reduce the number of code bits.

The Berger code [3] is the most frequently used code for the class of universal separable codes, while the (m,n)-code [4] is the most frequently used code for the class of nonseparable codes. In 1984 a context-oriented, separable code was proposed [5]. This code produces a decrease in the number of check bits by comparison with the Berger code, retaining the detectability of the code without any change. These codes are all optimal, i.e., each code requires the least number of additional bits to encode all the codes in its class.

The basic drawback of the Berger code is the substantial increase in the length of the word which it entails. In general, the use of words of lesser length as well as the use of sufficiently simple checkers make it possible to employ (m, n)-codes. The principal drawback of (m, n)-codes lies in the need to recode the data bits. This drawback becomes especially significant in those cases in which the method used to encode words that are protected against noise is defined by the particular application and cannot be arbitrarily varied even for the purpose of increasing noise immunity. For example, such a situation is encountered in the design of controllers based on the principle of microprogramming control [6]. In such controllers each bit of the output word is interpreted as a signal that initiates a definite action (microoperation) in the operational control device. It is precisely these types of devices which are considered in the present article.

Besides the features that have been noted, what is essential for our discussion is that in many cases the total number k of microoperations is quite high. At the same time, the number of microoperations that may be executed simultaneously and that are specified by a single control word (microinstruction) is much less than k. Thus, in each of the values of the controlled words employed most of the binary variables are equal to 0, and, moreover, the number of such values is much less than 2^k.

We finally have

$$r_1 = \varphi_1 \varphi_3 z_4 \vee \varphi_2 \varphi_4 z_4 \vee \varphi_1 \varphi_4 z_9 \vee \varphi_2 \varphi_3 z_9, \tag{20}$$

$$r_2 = \varphi_1 \varphi_3 z_9 \vee \varphi_2 \varphi_4 z_9 \vee \varphi_1 \varphi_4 z_4 \vee \varphi_2 \varphi_3 z_4. \tag{21}$$

Note that the circuits are testable on the set of admissible words and that no inverters are used in the circuits.

To calculate the conjunction $p = a\&b$ we represent each of the variables p, a, and b in paraphase code by means of pairs of signals (p_1, p_2), (a_1, a_2), and (b_1, b_2), respectively. Then the conjunction will be represented in the following form:

$$p_1 = a_1 \& b_1 \vee a_2 \& b_2, \quad p_2 = a_1 \& b_2 \vee a_2 \& b_1. \tag{22}$$

4. ESTIMATING THE COMPLEXITY OF THE CIRCUITS

To estimate the complexity of the control circuits we will use a realization in an FPGA basis. We will estimate the complexity of the circuits by the number of logic elements used in the particular circuit that realizes the look-up tables. For the sake of simplicity we will limit the discussion to the case in which the number of inputs in LUT $s = 4$. Suppose that k microoperations for the encoding are divided into m classes containing k_1, k_2,..., k_m elements, respectively. In this case the number of LUTs used in the circuit will be

$$H = 2\left(\sum_{i=1}^{m} \lceil (k_i - 1)/2 \rceil + m - 1\right). \tag{23}$$

Using the latter formula the following bounds may be obtained:

$$k + m - 2 \le H \le k + 2m - 1 \quad \text{or} \quad n - 2 \le H \le n + m - 2. \tag{24}$$

In the special case in which all the microoperations are incompatible and, consequently, $m = 1$ and $n = k + 1$, there is no circuit for use in computing a conjunction and, in general, the complexity of the checker satisfies the inequality $k - 1 \le H \le k$. Otherwise, in the case in which each microoperation forms a separate class, $m = k$ and $H = 2(k - 1)$. In automata belonging to this class, in which m is much less than k, the complexity of the control circuit is approximately proportional to k.

Experiments for benchmark circuits [10] with FPGA realization from the firm of Altera were undertaken. The results of the experiments are presented in Table 2. The number of LUTs required for realization of nonself-checking controllers are indicated in column F. The number of additional output variables required for the realization of a corresponding code on the controller outputs (for the Berger code, Smith code, and reduced (m, n)-code) is indicated in the respective ADD_Out columns. The number of LUTs required for realization of a self-checking system (controller + control circuit) with the use of the Berger, Smith, and reduced (m, n)-code is indicated in columns Γ_1, Γ_2, and Γ_3, respectively. Column Ω shows the relative reduction in hardware costs in supporting the property of self-checkability, achieved as a result of the application of the reduced (m, n)-code,

$$\Omega = \frac{(\min(\Gamma_1, \Gamma_2) - \Gamma_3) * 100}{\min(\Gamma_1, \Gamma_2)}. \tag{25}$$

The results of the experiments show that, on average, a greater number of additional output variables is required to realize the reduced (m, n)-code by comparison with the Berger and Smith codes, though at the same time there is still an approximately 30% reduction in hardware costs.

if and only if on all sets of arguments on which $R = 1$, r_1 , $r_2 = 01$ or 10, while $R = 0$ is represented by the combinations r_1 , $r_2 = 11$ or 00.

Obviously, the function may have several distinct representations in the paraphase code. To construct the control circuit we will use only those paraphase representations that do not require inverters upon realization.

Theorem 2. The function $F_{=1}(U)$ of variables of the set $U = \{u_1 , u_2 ,..., u_t\}$ admits a representation in paraphase code in the form

$$r_1 = F_{\geq 1}(U_1) \vee F_{\geq 2}(U_2) \quad \text{and} \quad r_2 = F_{\geq 1}(U_2) \vee F_{\geq 2}(U_1), \tag{10}$$
$$U_1, U_2 \subset U, \quad U_1 \cup U_2 = U, \quad U_1 \cap U_2 = \varnothing.$$

Proof. To prove the theorem the set of values of the input variables is divided into the following three groups:

(1) input variables equal to 0:

$$u_1 = u_2 = ,..., = u_t = 0, \quad F_{=1}(U) = 0, \text{ then } F_{\geq 1}(U_1) = F_{\geq 2}(U_2) = F_{\geq 1}(U_2) = F_{\geq 2}(U_1) = 0, \tag{11}$$

consequently, $r_1 = r_2 = 0$, which corresponds to the assertion of the theorem.

(2) precisely one of the t input variables is equal to 1, $F_{=1}(U) = 1$. Without loss of generality, we set $u_i = 1$ and $u_i \in U_1$. Then

$$F_{\geq 1}(U_1) = 1, \quad F_{\geq 2}(U_2) = F_{\geq 1}(U_2) = F_{\geq 2}(U_1) = 0, \tag{12}$$

consequently, $r_1 = 1$ and $r_2 = 0$, which corresponds to the assertion of the theorem.

(3) two or more input variables are equal to 1, $F_{=1}(U) = 0$. Suppose that all these unit variables belong to one of the subsets, say U_1. Then

$$F_{\geq 1}(U_1) = F_{\geq 2}(U_1) = 1, \quad F_{\geq 2}(U_2) = F_{\geq 1}(U_2) = 0, \tag{13}$$

consequently, $r_1 = r_2 = 1$, which corresponds to the assertion of the theorem. We now suppose that unit variables are contained in the each of the subsets U_1 and U_2. In this case,

$$F_{\geq 1}(U_1) = F_{\geq 1}(U_2) = 0, \tag{14}$$

consequently, $r_1 = r_2 = 1$, which also corresponds to the assertion of the theorem. The theorem is proved.

Let us illustrate Theorem 2 using as an example a realization of the function $F_{=1}(u_1, u_2, u_3, u_4, u_5)$. Let

$$U_1 = \{u_1, u_2\}, \quad U_2 = \{u_3, u_4, u_5\}, \tag{15}$$

whence

$$r_1 = u_1 \vee u_2 \vee u_3 u_4 \vee u_3 u_5 \vee u_4 u_5 \quad \text{and} \quad r_2 = u_1 u_2 \vee u_3 \vee u_4 \vee u_5 . \tag{16}$$

In the example from Table 1,

$$R = F_{=1}(z_1, z_2, z_5, z_7) \,\&\, F_{=1}(z_3, z_6, z_8) \,\&\, F_{=1}(z_4, z_9) . \tag{17}$$

The function $F_{=1}(z_1, z_2, z_5, z_7)$ is represented in paraphase code in the form of two functions,

$$\varphi_1 = z_1 \vee z_2 \vee z_5 z_7 \quad \text{and} \quad \varphi_2 = z_1 z_2 \vee z_5 \vee z_7 . \tag{18}$$

The function $F_{=1}(z_3, z_6, z_8)$ is represented in paraphase code in the form of two functions:

$$\varphi_3 = z_3 \vee z_6 \quad \text{and} \quad \varphi_4 = z_3 z_6 \vee z_8 . \tag{19}$$

The encoded output words are represented by columns y_1 , y_2 ,..., y_6 . To construct the code we use the following partition of set of microoperations into subsets of incompatible microoperations:

$$\{y_1, y_2, y_3, y_4, y_5, y_6\} = \{\{y_1, y_2, y_5\} \cup \{y_3, y_6\} \cup \{y_4\}\}. \quad (4)$$

The control bits in Table 1 are represented by the three rightmost columns. We introduce a composite enumeration of the data bits and control bits in the word thus encoded and denote a bit of this word by the symbol z_i, as was done in Table 1. In the set $Z = \{z_1, z_2, \ldots, z_n\}$ the partition

$$Z = \{Z_1, Z_2, \ldots, Z_m\}, \text{ where } Z_i = V_i \cup \{c_i\}, \quad i = 1, 2, \ldots, m \quad (5)$$

will correspond to the code.

In the example being considered here,

$$Z_1 = \{z_1, z_2, z_5, z_7\}, \; Z_2 = \{z_3, z_6, z_8\}, \; Z_3 = \{z_4, z_9\}. \quad (6)$$

Note that there is present in each code word precisely one variable from each subset Z_i . We call this partition the encoding partition. Several algorithms by means of which this partition may be obtained are known [7, 8].

It is best to use symmetric functions to describe the functional and structural organization of the control circuit [9]. We will use the symbol $F_h(U)$ to denote these functions, where U is the set of arguments specified by an enumeration or naming and h a condition that determines what is the number of arguments that must have unit value in order for the particular function to be equal to 1. For example, $F_{=1}(U)$ assumes unit value only in the case in which precisely one variable from U is equal to 1 (function "1-*out-of-n*").

Theorem 1. Let $Z = \{Z_1, Z_2, \ldots, Z_m\}$ is an encoding partition in which the compatibility relation is determined by the set W of admissible words. Then for any $Y_i \in W$,

$$R = F_{=1}(Z_1) \& F_{=1}(Z_2) \& \ldots \& F_{=1}(Z_m) = 1 \quad (7)$$

and for any unidirectional distortion Y_i, the function R is equal to 0.

Proof. The code is constructed so that precisely m bits in any code word will have unit value, one for each set Z_i. Thus, all the functions $F_{=1}(Z_i) = 1$, $i = 1, 2, \ldots, m$, and, consequently, $R = 1$. To prove the second part of the theorem, we will assume that as a result of unidirectional distortion, out of all the other bits that bit which corresponds to the variable $z_i \in Z_i$ will experience a change in its value. Obviously, with any change in the value of z_j (from 0 to 1 or from 1 to 0) the number of high bits belonging to the set Z_i will not be equal to 1 and this will hold true for any admissible distortions in the other bits that may occur at the same time as distortions in z_j . Consequently, $F_{=1}(Z_i) = 0$ and $R = 0$. Q.E.D.

By the theorem, the function (3) may be used to detect unidirectional errors.

3. SYNTHESIS OF CONTROL CIRCUIT FOR REDUCED (m, n)-CODE

As is done in most circuit designs of control circuits, we will represent the output signal R in the paraphase code by two signals r_1 and r_2 : if $r_1 \neq r_2$, there are no errors, otherwise an error has occurred either in the code circuit or in the control circuit. To implement the function (3) in the paraphase code, circuits that implement "1-out-of-n" functions and a conjunction must be constructed.

Definition 4. We will say that the function

$$R = F(U), \quad U = \{u_1, u_2, \ldots, u_t\} \quad (8)$$

admits a representation in paraphase code in the form

$$r_1 = f_1(U) \text{ and } r_2 = f_2(U), \quad (9)$$

In the present study we propose to construct self-checking controllers based on a *reduced* (m, n)-code that differ in two substantial respects from the traditional (m, n)-code. First, it is a separable code, i.e., the data bits do not change in the construction of the code. Control bits are added to the code so that the total number of unit components in the encoded word is equal to m. Second, the value of the control bits is selected so as to simplify the design of the control devices.

2. REDUCED (m, n)-CODE

We introduce the following notation:
y_i, microoperation or i-th bit of output word, $i = 1, 2,\ldots, k$; Y_i, microinstruction or output word of controller,

$$Y_j = \{y_{j_1}, y_{j_2},\ldots, y_{j_k}\},\ j = 1,2,\ldots,K; \tag{1}$$

V, set of microoperations,

$$V = \{y_1, y_2,\ldots, y_k\}; \tag{2}$$

W, set of microinstructions or set of (admissible) output words used,

$$W = \{Y_1, Y_2,\ldots,Y_K\}. \tag{3}$$

Definition 1. Two microoperations y_i and y_j are said to be compatible if they are encountered together in at least one microinstruction. Otherwise these microinstructions are incompatible.

Definition 2. The subset $V_i \subseteq V$ is called the subset of compatible (respectively, incompatible) microinstructions if any two microoperations belonging to this subset are compatible (respectively, incompatible).

Definition 3. Suppose that the set V is partitioned into subsets $V_1, V_2,\ldots, V_m$ of incompatible microinstructions such that

$$V = \{V_1 \cup V_2 \cup \ldots \cup V_m\} \text{ and } V_i \cap V_j = \varnothing, \text{ where } i, j \in \{1,2,\ldots,m\},\ i \neq j. \tag{3}$$

To construct the code we correlate to each subset V_i a single additional (control) bit c_i : $c_i = 1$ in those (and only those) microinstructions that do not contain any microoperations from V_i. We call such a code the *reduced (m,n)-code.*

Obviously in such an encoding each output word will contain precisely m high bits, one for each of the subsets V_i. An example of such an encoding is presented in Table 1.

Table 1

Microinstructions Encoded by Reduced (m, n)-Code

Microinstruction	Microoperations (data bits)						Control bits		
	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9
	y_1	y_2	y_3	y_4	y_5	y_6	c_1	c_2	c_3
Y_0	0	0	0	0	0	0	1	1	1
Y_1	1	0	1	1	0	0	0	0	0
Y_2	0	1	0	1	0	0	0	1	0
Y_3	0	0	0	1	1	1	0	0	0
Y_4	1	0	0	0	0	1	0	0	1
Y_5	0	0	0	1	0	1	1	0	0
Y_6	0	0	1	1	0	0	1	0	0
Y_7	0	0	0	0	1	0	0	1	1
Y_8	0	0	1	0	0	0	1	0	1
Y_9	0	1	0	0	0	1	0	0	1

Table 2

Results of Experiments

N	Example	Φ	Berger Code		Smith Code		Reduced (m, n)-Code		Ω (%)
			Add_Out	Γ_1	Add_Out	Γ_2	Add_Out	Γ_3	
1	CSE	79	3	39	2	39	3	38	2.6
2	EX1	206	5	165	4	118	9	30	74.6
3	EX6	54	4	32	3	24	6	16	33.3
4	PLANET	182	5	166	3	156	10	74	52.6
5	PMA	112	4	62	3	56	7	46	17.9
6	S386	58	3	33	2	29	3	24	17.2
7	S820	263	5	62	2	37	3	25	32.4
8	S832	261	5	87	2	50	3	35	30
9	SAND	192	4	114	3	95	5	80	15.8
10	SSE	52	3	26	2	27	3	27	0
	Average	145.9	4.1	78.6	2.6	63.1	5.2	39.5	27.6

5. CONCLUSION

In the present study a method for the design of self-checking controllers is proposed. A reduced (m, n)-code is used to support the properties of self-checkability. The reduced (m, n)-code is, in the general case, not optimal, though because of some redundancy in the encoding of the output words, the hardware costs required to detect any unidirectional errors in an output word can be reduced. The results of the experiments on benchmark circuits show that in practically all cases application of the reduced (m, n)-code leads to simpler circuit designs. Moreover, the reduction in hardware costs in most complex automata reaches 50%. Because of this result, it may be suggested that the proposed method of constructing self-checking controllers may be of interest for practical application.

REFERENCES

[1] P. Lala, Self-Checking and Fault-Tolerant Digital Design, Morgan Kaufmann, San Francisco, 2000.
[2] D.K. Pradhan and J.I. Stiffler, "Error correcting codes and self-checking circuits in fault tolerant computers," Computer, pp. 27–37, March, 1980.
[3] J.M. Berger, "A note on error detection codes for completely asymmetric binary channels," Inform. Contr., vol. 4, pp. 68–73, March, 1961.
[4] C.V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," Inform. Contr., vol. 5, pp. 64–71, March, 1962.
[5] J. Smith, "On separable unordered codes," IEEE Trans. Computers, vol. C-33, no. 8, pp. 741–743, August, 1984.
[6] S. Baranov, Logic Synthesis for Control Automata, Kluwer Academic, Dordrecht/Boston, 1994.
[7] M.C. Paull and S.H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," IRE Trans. Electron. Comput., EC-8.3, 1959.
[8] V.I. Ostrovskii and Yu.V. Pottosin, "Investigation of algorithms for use in the search for maximum complete subgraphs in a symmetric graph," Avtomatika i Vychislitel'naya Tekhnika, no. 2, pp. 19–26, 1970.
[9] S.B. Akers, "A rectangular logic array," IEEE Trans. Computers, vol. C-21, pp. 848–857, August, 1972.
[10] R. Lisanke, "Logic synthesis benchmark circuits," Intern. Workshop on Logic Synthesis, Research Triangle Park, NC, May, 1989.

Received following revisions 17 June 2002; originally submitted 13 May 2002