Cybernetics and Systems Analysis, Vol. 49, No. 1, January, 2013

NEW MEANS OF CYBERNETICS, INFORMATICS, COMPUTER ENGINEERING, AND SYSTEMS ANALYSIS

MODIFICATION OF THE MICROCOMMAND ADDRESSING SYSTEM IN A CONTROL UNIT WITH CODE SHARING

A. A. Barkalov,[a] L. A. Titarenko,[a] and A. N. Miroshkin[b]

UDC 004.3

Abstract. *This article proposes two modifications of the microcommand addressing system in a compositional microprogram control unit with code sharing. The modifications are based on using FSM pseudoequivalent states to reduce the number of rows in the FSM transition table and thereby to reduce the complexity of the combinational part of the device. Methods are proposed for synthesizing compositional control units with a modified microcommand addressing system. The research results are presented and appropriate fields of application of the methods proposed are considered.*

Keywords: *graph-scheme of an algorithm, control unit, compositional microprogram control unit, code sharing, FPGA programmable logic, microcommand addressing.*

INTRODUCTION

A digital system consists of a control unit (CU) and an operational unit. The CU produces a sequence of control signals under the action of which the operational unit executes definite microoperations [1]. To specify a sequence of actions, the language of graph-schemes of algorithms (GSAs) is used. The functioning of a CU implementing a GSA is described by a digital FSM model [2]. Practical implementations of CUs of different types are characterized by factors such as hardware expenditures and time characteristics (clock cycle duration and time to failure). In practice, as a rule, the problem of minimization of hardware expenditures is topical [3] whose solution depends on distinctive features of a control algorithm and the element basis used for implementing a device. In the case when a linear algorithm should be implemented, it is expedient to use the model of a compositional microprogram control unit (CMCU) from [4, 5].

This work is devoted to the investigation of the process of implementation of a CMCU circuit in the basis of integral circuits of the type of FPGAs (field-programmable gate arrays) [6, 7]. The functional element of an FPGA is the so-called look-up table (LUT). Each LUT element can be used not only as a functional generator but also as a synchronous small memory. In addition to LUT elements, the majority of modern FPGAs contain special programmable memory blocks each of which is a synchronous random access memory (RAM) whose size equals 18 Kb [8]. A block RAM is used in circuits to store a relatively larger amount of data more efficiently than the mentioned distributed memory based on LUT elements. An implementation of finite-state machines (FSMs) with FPGA-based memory is described [9–13].

Some approaches to modifying a microcommand addressing system on the basis of the model of a CMCU with code sharing [4] are considered below, but the proposed idea can also be implemented on the basis of other structures of control units with memory.

[a]University of Zielona Góra, Zielona Góra, Poland, *A.Barkalov@iie.uz.zgora.pl*. [b]Donetsk National Technical University, Donetsk, Ukraine. Translated from Kibernetika i Sistemnyi Analiz, No. 1, January–February, 2013, pp. 161–171. Original article submitted October 13, 2011. Updated article submitted March 12, 2012.

1060-0396/13/4901-0139

MODEL OF A CMCU WITH CODE SHARING

The initial data for synthesizing a CU consist of the graph-scheme of a control algorithm. Let an arbitrary GSA Γ consist of a set of vertices B and arcs (edges) E connecting these vertices. In this case, $B = \{b_0, b_E\} \cup B_1 \cup B_2$, where b_0 and b_E are the initial and terminal vertices of the GSA and B_1 and B_2 are the sets of operator and conditional vertices, respectively. Vertices $b_m \in B_1$ contain collections of microoperations $Y(b_m) \subseteq Y$, where $m = \overline{1, M}$, $M = |B_1|$, is the total number of operator vertices of the GSA and $Y = \{y_1, \ldots, y_N\}$ is the set of microoperations (output functions of the FSM). Vertices $b_q \in B_2$ contain elements of a set of logical conditions $X = \{x_1, \ldots, x_L\}$. We introduce some definitions used below.

Definition 1. A linear chain α_g of operators of a GSA Γ is called a finite sequence of operator vertices $\langle b_{g_1}, \ldots, b_{g_{F_g}} \rangle$ such that, for any pair of its adjacent components, it contains an arc $\langle b_{g_i}, b_{g_{i+1}} \rangle \in E$, where $i = \overline{1, F_g - 1}$ and F_g is the number of components in linear operator chains (LOCs) α_g.

Definition 2. An operator vertex $b_m \in B^g$, where $m = \overline{1, F_g}$ and $B^g \subseteq B_1$ is the set of operator vertices belonging to a LOC α_g is called its input if there is an arc $\langle b_t, b_m \rangle \in E$, where $b_t \notin B^g$. Each LOC α_g has an arbitrary number of inputs forming a set $I(\alpha_g) = \{I_g^1, I_g^2, \ldots\}$.

Definition 3. An operator vertex $b_m \in B^g$ is called an output of a LOC α_g if there is an arc $\langle b_m, b_t \rangle \in E$, where $b_t \notin B^g$. An arbitrary LOC α_g has only one output denoted by O_g.

Definition 4. Chains α_i and α_j are called pseudoequivalent LOCs (PLOCs) if there are arcs $\langle b_i, b_t \rangle \in E$ and $\langle b_j, b_t \rangle \in E$, where b_i and b_j are the outputs of the LOCs α_i and α_j, respectively.

Definition 5. A graph-scheme Γ of an algorithm is called linear if the following condition is satisfied:

$$\frac{M}{G} \geq 2, \tag{1}$$

where G is the number of linear operator chains of the GSA, i.e., such a GSA is linear if the number of operator vertices of the algorithm exceeds a minimum number of chains by a factor of no less than 2. Under condition (1), it makes sense to use the CMCU model [4] to implement a control algorithm.

To find a partition of the set B_1 of operator vertices into a minimum number of LOCs, the method described in [4] is used. After the formation of the set $C = \{\alpha_1, \ldots, \alpha_G\}$, each LOC α_g contains F_g components, a binary code $K(\alpha_g)$ of length

$$R_1 = \lceil \log_2 G \rceil \tag{2}$$

is assigned to it, and a code $K(b_{g_i})$ of length

$$R_2 = \lceil \log_2 (F_{\max}) \rceil \tag{3}$$

is assigned to each component b_{g_i}, where $F_{\max} = \max(F_1, \ldots, F_G)$ is the number of components in the LOC of maximal length. For coding LOCs, elements of a set τ are used and, for coding LOC components, elements of a set T are used, where $|\tau| = R_1$ and $|T| = R_2$.

Components are coded in natural order, i.e.,

$$K(b_{g_{i+1}}) = K(b_{g_i}) + 1 \quad (g = \overline{1, G};\ i = \overline{1, F_g}). \tag{4}$$

An operator vertex b_m corresponds to the microcommand MI_m whose address $A(MI_m)$ is specified as follows:

$$A(MI_m) = K(\alpha_g) * K(b_{g_i}), \tag{5}$$

where $*$ is the concatenation sign and the vertex b_m corresponds to the component b_{g_i} of the LOC α_g.

The application of the principle of code sharing means that the address formation circuit (AFC) implements the following system of excitation functions of the counter CT and register RG:

$$\begin{aligned} \Phi &= \Phi(\tau, X); \\ \Psi &= \Psi(\tau, X). \end{aligned} \tag{6}$$

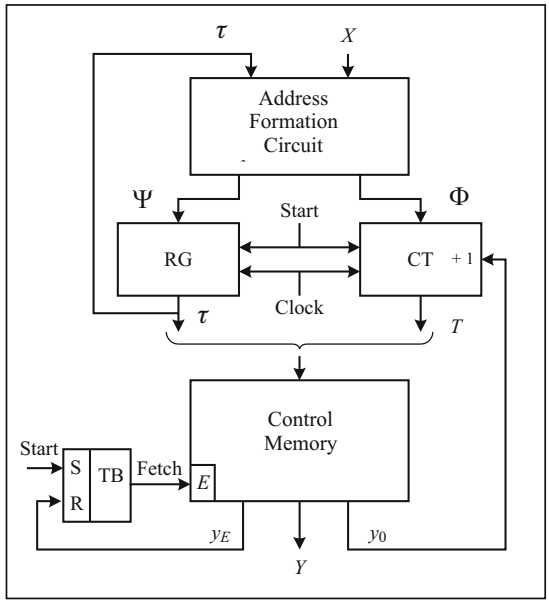

Fig. 1. Block diagram of a CMCU with code sharing.

The CMCU model with code sharing (a CS-structure) is presented in Fig. 1 and functions as follows. At the beginning of operation initiated by the signal Start, the trigger TB that enables reading from control memory (CM) is switched to logic "1." The address formation circuit provides the transition to the input vertex of one of LOCs α_g. The address formed for it and stored in the counter participates in fetching a definite microcommand from memory. If the current vertex is not a LOC output, then the signal y_0 is formed that allows the counter to increase its content and thereby to address the next component of the LOC α_g according to the rule of natural microcommand addressing (4). After achieving the output O_g of the LOC α_g, the signal y_0 is not formed, which leads to assigning the address of transition to an input vertex of the next LOC to the counter. This address is formed by the AFC from the code of the current LOC (this code is stored in the register RG). The device operation terminates when the signal y_E is formed that inhibits the further reading of microcommands. This signal is added to all the vertices whose outputs are connected with the GSA terminal vertex.

In implementing a CMCU circuit in the FPGA basis, in addition to CM, its blocks are also constructed from LUT elements and distributed memory elements (latched triggers) and, in implementing its memory, embedded memory blocks are used. If some part of resources of embedded memory blocks is not used, they can be used for decreasing the number of LUT elements in the addressing circuit, which reduces hardware expenditures for the implementation of the device and improves time characteristics of the circuit obtained.

MAIN IDEA OF THE PROPOSED METHOD

Transitions from pseudoequivalent chains are presented in the CMCU transition table by rows differing only in the source of the code for the AFC. Thus, they can be replaced by one row with the code of a class of PLOCs as the source and, as a result, the number of table rows and also the number of arguments and terms in functions of system (6) decrease. Codes of PLOC classes can be stored in free resources of embedded memory blocks of FPGA microcircuits. In this article, the following two approaches to modifying the microcommand addressing system are proposed: an extension of the format of microcommands, i.e., the addition an auxiliary field to the format that will contain the code of a PLOC class; a modification of chains, i.e., the inclusion of an additional vertex with the code of a PLOC class in the initial GSA at the end of each linear operator chain that is not connected with the terminal vertex.

Let a LOC $\alpha_g \in C_1$ if its output O_g is not connected with the terminal vertex b_E. We find a partition $\Pi_C = \{B_1, \ldots, B_I\}$ of the set C_1 into PLOC classes and code each of them by a binary code of length $R_3 = \lceil \log_2 I \rceil$. For coding PLOC classes, we use variables from a set $Z = \{z_1, \ldots, z_{R_3}\}$. Then system (6) is transformed to the form

$$\Phi = \Phi(Z, X); \quad \Psi = \Psi(Z, X). \tag{7}$$

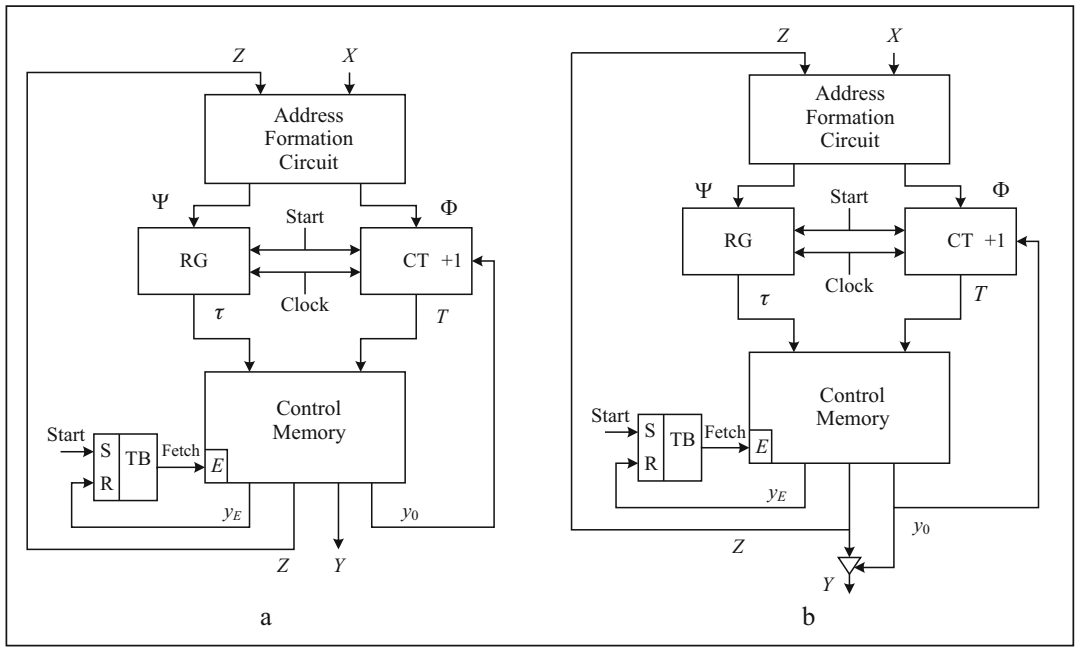

Fig. 2. Block diagrams of CMCUs with the following modifications in the microcommand addressing system: (a) with an extension of the format of microcommands and (b) with a modification in the GSA.

With extending the format of microcommands, their number in memory equals the following corresponding value for a CMCU with code sharing:

$$M_1 = (G-1)\cdot 2^{\lceil \log_2 |\alpha_{\max}| \rceil} + |\alpha_{\min}|,$$

where $\alpha_{\max}$ and $\alpha_{\min}$ are chains with maximum and minimum numbers of components, respectively. For ensuring a minimum number of microprogram words, the maximal code out of the LOC codes being used should be assigned to the LOC $\alpha_{\min}$, and, as a result, the corresponding memory content will be stored in the memory space with maximal addresses.

After adding the auxiliary field, the length of each word will be equal to $N_1 = N_Y + 2 + R_3$, where N_Y is the number of bits required for coding the set of microcommands $y_i \in Y$ (in the case of unitary coding of microoperations, we have $N_Y = |Y|$), and the constant 2 reserves two bits for storing the internal variables y_0 and y_E. The lengths of the register and counter are equal to the values specified by formulas (2) and (3).

A modification in a LOC can increase the number of microprogram words up to the value of $M_2 = (G-1)\cdot 2^{\lceil \log_2(|\alpha_{\max}|+1) \rceil} + |\alpha_{\min}| + 1$.

In a trivial implementation of the proposed approach, the word length is found by the formula $N_2 = \max(N_Y; R_3) + 2$.

The lengths of the register and counter after modifying chains are specified by formulas (2) and (3). However, the value of the parameter $F_{\max}$ can be increased, which will lead to the increase in the number of bits in the register. Note that, with increasing the counter length, the SFA will form R_2 variables as before. The reason is that the direct jump to a microcommand that contains only the code of a PLOC class will never take place. In this case, the high-order bit of data for the counter will always be equal to zero.

We make the convention that a microprogram can be realized with parameters M_i and N_i using one block of built-in memory.

We denote by FCS and MCS the CMCU models presented in Figs. 2a and 2b, respectively.

Principles of functioning of the basic model of a CMCU with code sharing (see Fig. 1) and the FCS and MCS models are similar. The proposed method of synthesizing the considered structures includes the following stages:

— formation of the sets C, C_1, and Π_C for the GSA Γ;

— optimal coding of chains $\alpha_g \in C_1$ and their components;

— coding of classes of pseudoequivalent chains $B_i \in \Pi_C$;

— formation of the CM content;

— formation of the CMCU transition table and system of transition functions (7);

— synthesis of the CU circuit in a given basis.

The first stage is carried out according to the method described in [4]; in this case, the number G of LOCs $\alpha_g \in C$ is minimally possible. Π_C is trivially partitioned on the basis of definition 4.

An optimal coding of a LOC $\alpha_g \in C_1$ is performed according to a method similar to the optimal coding of states of a Moore FSM [15]. To solve this problem, well-known methods of symbolic minimization are used [3]. Components of all LOCs are trivially coded as follows: the codes whose decimal equivalents are equal to zero, one, two, etc. are assigned, respectively, to the first, second, etc. components. This coding satisfies equality (4).

The method of coding classes $B_i \in \Pi_C$ must minimize the number of terms in a function $z_r \in Z$. Adapting the well-known (for D-triggers) method of coding states [15], we obtain that the more generalized intervals are necessary for the representation of a class $B_i \in \Pi_C$, the smaller number of 1s must contain its code.

The CM content is created in the form of a table with the fields $A(MI_m)$, $Y(b_m)$, y_0, y_E, and $C(B_i)$. In this case, for the FCS CMCU structure, the fields $Y(b_m)$ and $C(B_i)$ are separate columns as well as the other fields. In the case of the MCS structure, these fields are written in one column whose content is treated depending on the value of the variable y_0. In this case, the column width is selected as $\max(N_Y; R_3)$, and nonsignificant zeros are added to codes in fields with smaller width.

Before the construction of the memory content, the initial GSA is transformed [4]. If a vertex b_{g_i} of a LOC $\alpha_g \in C_1$ is not the output of the LOC, then the variable $y_0 = 1$ is inserted into it and determines the natural addressing for the next component of the LOC. If $\alpha_g \notin C_1$, then the variable $y_E = 1$ is inserted into the vertex O_g and completes the operation of the FSM.

The transition table for the CMCUs being considered contains the following columns: B_i is the PLOC class from which a transition is performed, $k(B_i)$ is the code of the PLOC class, b_m is the vertex to which the transition is performed, $A(MI_m)$ is the address of the microcommand corresponding to the vertex b_m, X_h is the conjunction of logical variables determining the transition from the output O_g of a LOC $\alpha_g \in B_i$ to the vertex b_m in the hth row of the transition table, Ψ_h and Φ_h are the excitation functions of the register and counter, respectively, and h is the number of a term (a transition table row).

Each function of system (7) is represented in the form

$$d_i = \bigvee_{h=1}^{H} \left(P_i^h \wedge B_h \wedge \left(\bigwedge_{l=1}^{L} C_l^h x_l^h \right) \right) \quad (i = \overline{1, R}), \tag{8}$$

where d_i is the excitation function of the ith bit of the FSM state register ($d_i = \tau_i$ when $i = \overline{1, R_2}$ and $d_{i+R_2} = T_i$ when $i = \overline{1, R_1}$); P_i^h is a variable determining the presence of the function d_i in the hth row of the transition table ($P_i^h = 1$ if the function d_i is present in the hth row and $P_i^h = 0$ otherwise); B_h is the conjunction of bits q_r^h of the code of a PLOC class (q_r^h is the value of the rth bit ($r = \overline{1, R_3}$) of the code $k(B_i)$ of a PLOC class $B_i \in \Pi_C$ in the hth row of the transition table; $q_r^h = q_r$ if the rth bit of the code is equal to logic "1" and $q_r^h = \overline{q}_r$ otherwise); x_l^h is a variable determining the value of the logical condition x_l in the hth row of the transition table ($x_l^h = x_l$ if the transition in the hth row takes place when the condition is fulfilled, and $x_l^h = \overline{x}_l$ when the condition is not fulfilled); C_l^h a variable determining the presence of the logical condition x_l in the hth row of the transition table ($C_l^h = 1$ when the signal x_l is present and $C_l^h = 0$ when it is absent), H is the number of rows in the transition table, R_3 is the length of the code of a PLOC class, and L is the number of logical conditions.

The synthesis of a CMCU circuit is reduced to the implementation of the model of its control unit in the FPGA basis with the help of one of standard applied packages [16, 17] and is beyond the scope of this article.

TABLE 1

T_2T_1	Codes of Vertices for the LOC ($\tau_3\tau_2\tau_1$)							
	α_0 (000)	α_1 (001)	α_2 (010)	α_3 (011)	α_4 (100)	α_5 (101)	α_6 (110)	α_7 (111)
00	b_0	b_3	b_6	b_8	b_{11}	b_{13}	b_{15}	b_{17}
01	b_1	b_4	b_7	b_9	b_{12}	b_{14}	b_{16}	—
10	b_2	b_5	(b_{20})	b_{10}	(b_{22})	(b_{23})	(b_{24})	—
11	(b_{18})	(b_{19})	—	(b_{21})	—	—	—	—

a

Address		Microcommand										b_m	LOC α_i
$\tau_3\ \tau_2\ \tau_1$	$T_2\ T_1$	y_E	y_1	y_2	y_3	y_4	y_5	y_6	y_0	z_2	z_1		
0 0 1	0 0	0	1	0	1	0	0	0	1	0	0	b_3	α_1
	0 1	0	0	0	0	1	0	0	1	0	0	b_4	
	1 0	0	0	0	0	0	1	0	0	**0**	**1**	b_5	
	1 1	0	0	0	0	0	0	0	0	0	0	—	
0 1 0	0 0	0	0	1	0	0	0	0	1	0	0	b_6	α_2
	0 1	0	0	0	1	0	0	1	0	**0**	**1**	b_7	
	1 0	0	0	0	0	0	0	0	0	0	0	—	
	1 1	0	0	0	0	0	0	0	0	0	0	—	

b

Address		Microcommand								b_m	LOC α_i
$\tau_3\ \tau_2\ \tau_1$	$T_2\ T_1$	y_E	y_1 (z_2)	y_2 (z_1)	y_3	y_4	y_5	y_6	y_0		
0 0 1	0 0	0	1	0	1	0	0	0	1	b_3	α_1
	0 1	0	0	0	0	1	0	0	1	b_4	
	1 0	0	0	0	0	0	1	0	1	b_5	
	1 1	0	**0**	**1**	0	0	0	0	0	b_{19}	
0 1 0	0 0	0	0	1	0	0	0	0	1	b_6	α_2
	0 1	0	0	0	1	0	0	1	1	b_7	
	1 0	0	**0**	**1**	0	0	0	0	0	b_{20}	
	1 1	0	0	0	0	0	0	0	0	—	

Fig. 3. Fragments of the contents of control memory for the CMCU with an extension of the format of microcommand (a) and with a modification of LOCs of the control algorithm (b).

EXAMPLE OF APPLICATION OF THE PROPOSED METHOD

Let some GSA Γ_1 contain $M = 17$ operator vertices that form the set $C = \{\alpha_0, \ldots, \alpha_7\}$ of LOCs, where $\alpha_0 = \langle b_0, b_1, b_2 \rangle$, $\alpha_1 = \langle b_3, b_4, b_5 \rangle$, $\alpha_2 = \langle b_6, b_7 \rangle$, $\alpha_3 = \langle b_8, b_9, b_{10} \rangle$, $\alpha_4 = \langle b_{11}, b_{12} \rangle$, $\alpha_5 = \langle b_{13}, b_{14} \rangle$, $\alpha_6 = \langle b_{15}, b_{16} \rangle$, and $\alpha_7 = \langle b_{17}, b_E \rangle$. $G = 8$ LOCs out of $M = 17$ operator vertices are obtained. Condition (1) is fulfilled and, hence, it is expedient to use the CMCU model.

The chain $\alpha_7 \notin C_1$ since it contains the terminal vertex of the GSA. The set of logical conditions includes $L = 4$ elements, and the FSM produces $N = 6$ control signals. The partition $\Pi_C = \{B_0, \ldots, B_3\}$ into PLOC classes is obtained from the set C_1, where $B_0 = \{\alpha_0, \alpha_5\}$, $B_1 = \{\alpha_1, \alpha_2, \alpha_4\}$, $B_2 = \{\alpha_3\}$, and $B_3 = \{\alpha_6\}$.

To code the GSA vertices according to formulas (2) and (3), it suffices to use $R_1 = 3$ elements of the set $\tau = \{\tau_1, \tau_2, \tau_3\}$ and $R_2 = 2$ elements of the set $T = \{T_1, T_2\}$. The codes of vertices of the GSA Γ_1 are presented in Table 1 in which the added vertices for the CMCU with modifications in LOCs are given in brackets. According to expression (5), the code of a vertex is formed as the concatenation of the LOC code and the code of the component itself, i.e., $k(b_0) = 00000$, $k(b_3) = 00100$, $k(b_{16}) = 11001$, etc. We code classes $B_i \in \Pi_C$ as follows: $k(B_0) = 00$, $k(B_1) = 01$, $k(B_2) = 10$, and $k(B_3) = 11$. In synthesizing the CMCU with modified chains of the control algorithm, the vertex containing the code $k(B_i)$ is introduced at the end of each LOC $\alpha_g \in C_1$, where $\alpha_g \in B_i$.

Recall that the variable y_0 is written in vertices that are not outputs of LOCs $\alpha_g \in C_1$. The variable y_E is written in the vertices connected with the terminal vertex of the algorithm. The other fields of the set Y are formed according to the GSA content. Fragments of memory content for the CMCU structures being investigated are presented in Fig. 3, where the black font highlights PLOC codes participating in the formation of the address of a transition, and the light font highlights spare memory areas. Contents of all the other chains $\alpha_g \in C$ are similarly obtained.

The transition table is constructed from the initial graph-scheme, and functions (7) are obtained from this table. Let us consider the transition table for the GSA Γ_1 (Table 2).

TABLE 2

B_i	$k(B_i)$	b_m	$A(MI_m)$	X_h	Ψ_h	Φ_h	h
B_0	00	b_3	00100	x_1	τ_1	—	1
		b_8	01100	$\overline{x_1}x_2$	$\tau_2\tau_1$	—	2
		b_6	01000	$\overline{x_1}\,\overline{x_2}$	τ_2	—	3
B_1	01	b_{15}	11000	x_4	$\tau_3\tau_2$	—	4
		b_{17}	11100	$\overline{x_4}$	$\tau_3\tau_2\tau_1$	—	5
B_2	10	b_{11}	10000	x_3	τ_3	—	6
		b_{13}	10100	$\overline{x_3}$	$\tau_3\tau_1$	—	7
B_3	11	b_5	00110	1	τ_1	T_2	8

We represent each function of system (7) in the form of functions (8) as follows:

$$B_0 = \overline{z_1}\,\overline{z_0};\ B_1 = \overline{z_1}z_0;\ B_2 = z_1\overline{z_0};\ B_3 = z_1z_0;$$

$$\tau_3 = B_1 \vee B_2;\ \tau_2 = B_0\overline{x_1} \vee B_1;$$

$$\tau_1 = B_0x_1 \vee B_0\overline{x_1}x_2 \vee B_1\overline{x_4} \vee B_2\overline{x_3} \vee B_3;\ T_2 = B_3;\ T_1 = 0.$$

As has been already mentioned, the stage of implementation of CMCU circuits is not considered in this article. However, the authors realized a computer-aided design system (CADS) that, together with the package Xilinx ISE Webpack, allows one to synthesize the proposed CMCU circuits.

RESULTS OF INVESTIGATIONS

An analysis was conducted with the help of the developed CADS for control units that synthesizes models of structures of a control unit from the description of the graph-scheme of the control algorithm in the XML format. A model is understood to be a VHDL description of the circuit of a device and files for programming its ROM (for devices with memory). These models are entered into the system Xilinx ISE Webpack which implements them in terms of one of available microcircuits. The implementation project files of a device contain data on the involved hardware resources of a microcircuit, critical paths of a signal and time parameters of the device, its power consumption, etc.

For the experiment being considered, GSAs with the following parameters are chosen: from 10 to 500 vertices with the step equal to 10; from 50% to 90% of operator vertices with the step equal to 10%; 15 microoperations; five logical conditions.

For each GSA, the structure of aCMCU with code sharing (CS) and also the FCS- and MCS-structures being investigated were synthesized. With a view to comparing with the class of FSMs with hardwired logic, Mealy FSMs were also synthesized. Each measurement presented in the diagrams is the average value of the results of synthesizing five different GSAs with identical parameters.

An analysis of hardware expenditures has shown the efficiency of using the proposed methods for all the investigated GSAs (Fig. 4).

The proposed CMCU structures (see Fig. 4) require smaller hardware expenditures than the basic CMCU with code sharing and a Mealy FSM. Note that hardware expenditures for an implementation of a Mealy FSM increase with increasing the percentage of operator vertices in a GSA but decrease for an implementation of a device of the CMCU class. The differences between the values of hardware expenditures for FCS- and MCS-structures of CMCUs are minimal and resemble statistical deviations. For the further comparison of data structures, we will investigate time characteristics of the obtained circuits.

The time characteristics being investigated are the clock signal period and duration of formation of output functions. The first parameter makes sense when logical conditions are specified depending on the operating mode. If the choice of an algorithm branch is determined by the result of the previous operation, then the second time parameter makes sense as determining the maximal time period after the formation of the last condition out of the logical conditions during which the device generates output functions.

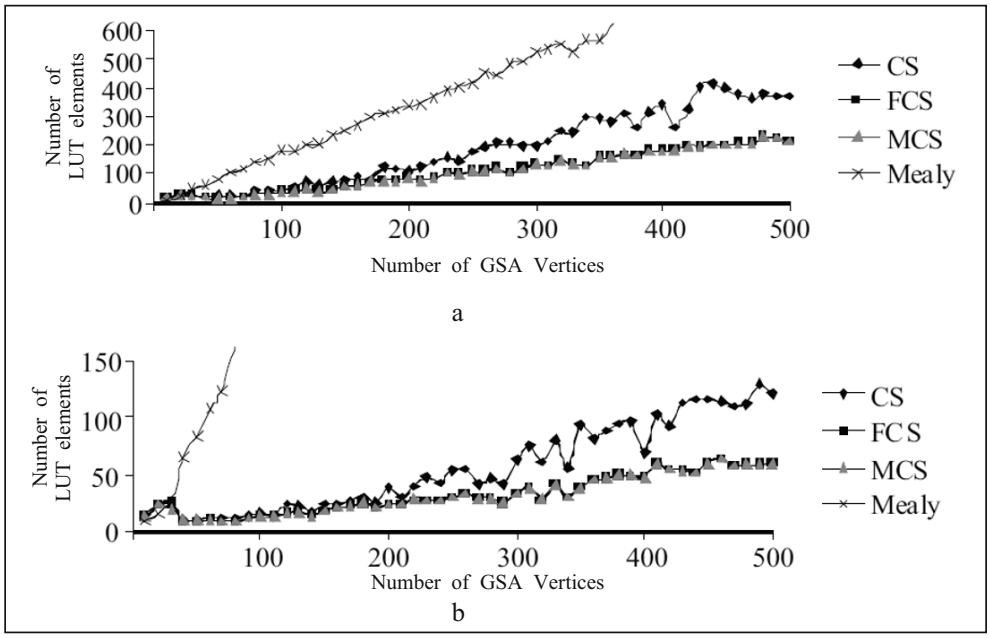

Fig. 4. Diagrams of hardware expenditures in implementing different CMCU structures when the GSA contains 70% of operator vertices (a) and 90% of operator vertices (b).

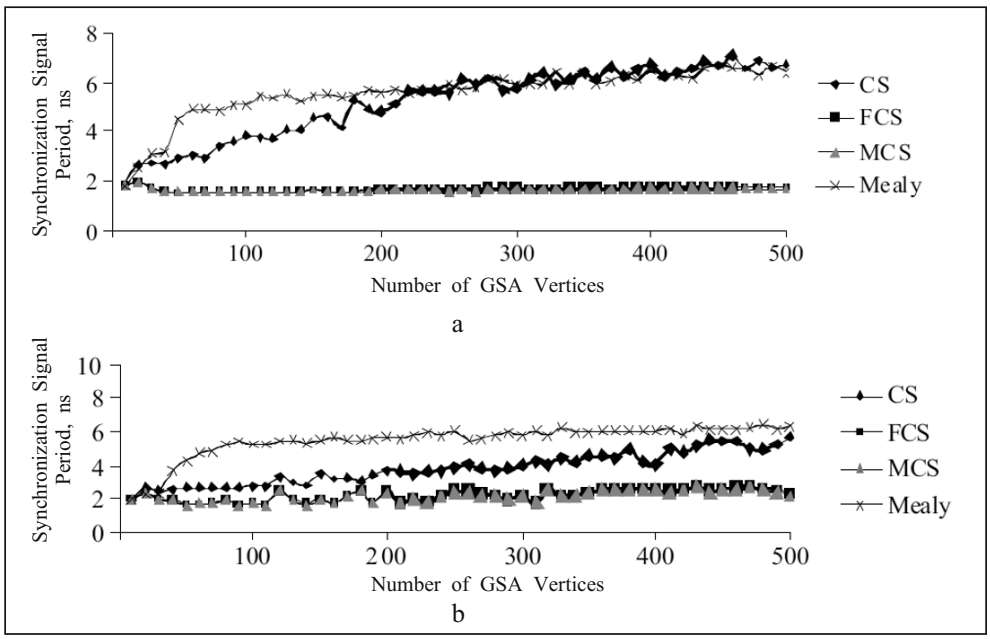

Fig. 5. Diagrams of the synchronization signal period of circuits of different control units when the GSA contains 70% of operator vertices (a) and 90% of operator vertices (b).

An analysis of the diagrams presented in Fig. 5 allows one to draw the conclusion that the proposed CMCU structures not only occupy a smaller area but also can have a smaller cycle time than CMCUs with code sharing and a Mealy FSM. However, the comparison of the synchronization signal period, as before, does not allows one to draw the conclusion about some preferable control unit structure.

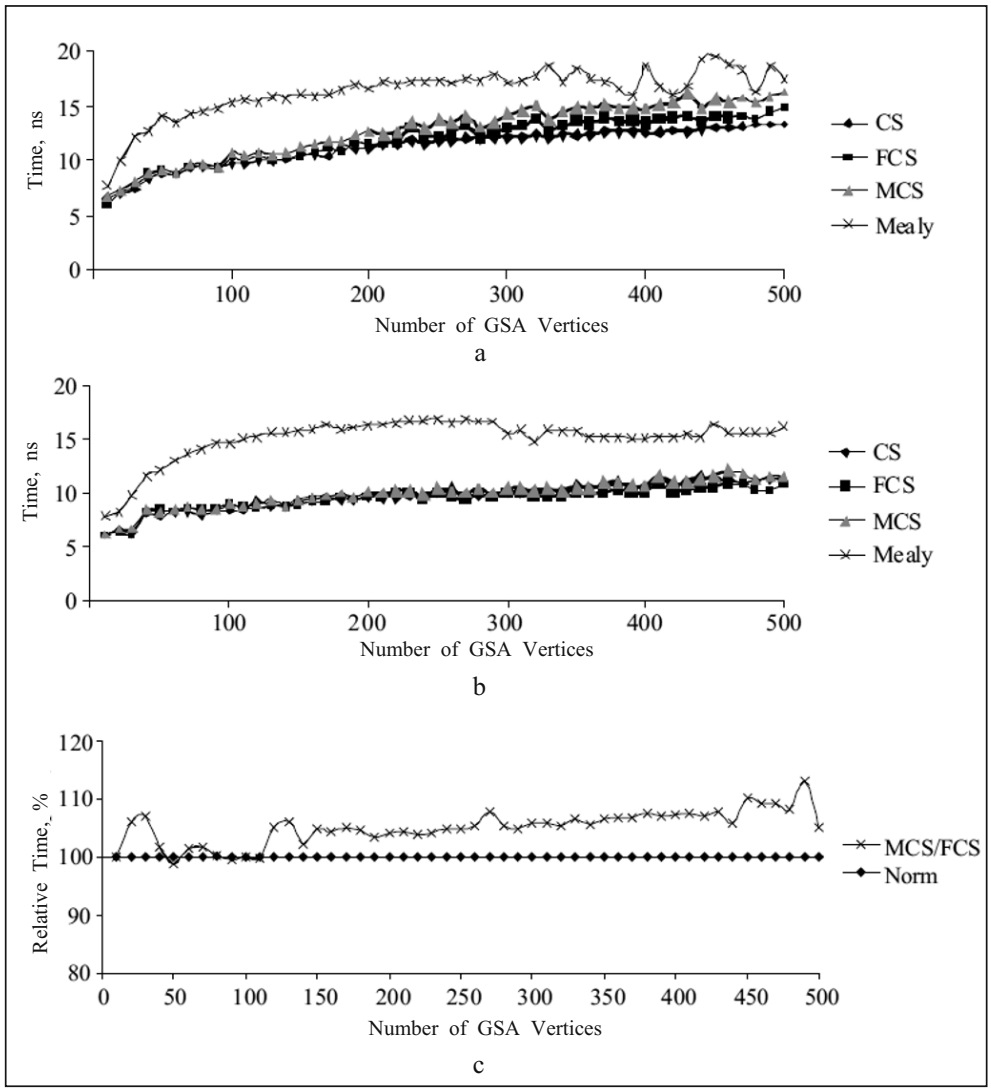

Fig. 6. Diagrams of duration of forming CU output functions when a GSA contains 70% of operator vertices (a), 90% of operator vertices (b), and also diagrams of relative durations of forming output functions in the case when a GSA contains 90% of operator vertices (the values for the MCS-structure are normalized with respect to the values for the FCS-structure) (c).

The performed investigations of the duration of forming output functions (Fig. 6) have shown that the CMCU structure with the extension of the microcommand format forms output functions quicker than the CMCU with a modification in LOCs of the control algorithm for all the considered GSAs. This is conditioned by the additional circuit at the output of the device; during the moments of transition of the FSM from one LOC to another, this circuit makes it impossible to treat the code of a PLOC class as output signals.

Note that all the obtained circuits of control units with memory used no more than one block of an FPGA microcircuit with built-in memory.

CONCLUSIONS

An extension of the format of microcommands and a modification in LOCs of the initial GSAs have shown a decrease in hardware expenditures by 40% on the average in comparison with the basic CMCU structure in implementing CU circuits in the basis of modern FPGA microcircuits.

To implement the proposed CMCU structures in the basis of the Spartan-3 microcircuit of Xilinx, Inc., the authors obtain an analytical dependence between parameters of an input GSA and hardware expenditures necessary for implementing a device. The dependence is of the form $Q = (-0.026N^2 + 2.56N - 10.11)p$, where N is the total number of vertices in the GSA, and p is the portion of operator vertices in it, $p \in [0{,}5; 0{,}9]$. This dependence can be used for other FPGA microcircuits of Xilinx with four-input LUT elements. The error of the formula for a GSA in which the number of vertices is no more than 70 does not exceed 20%, whereas the statistical error can reach the value of hardware expenditures for small GSAs.

The duration of the clock signal period and time of formation of output signals for the proposed structures are approximately 1.7–2.5 ns in comparison with 5–6 ns for a Mealy FSM; for a CMCU, these values are fixed and depend on the type of a microcircuit, and, in a Mealy FSM, delays increase with increasing the complexity of a circuit. The use of a specialized system for computer-aided design of control units decreases the time of development of digital devices (from several hours to several minutes for unsophisticated GSAs).

REFERENCES

1. S. Baranov, Logic and System Design of Digital Systems, TUT Press, Tallinn (2008).
2. V. M. Glushkov, Synthesis of Digital Automata [in Russian], Fizmatgiz, Moscow (1962).
3. G. DeMicheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, N.Y. (1994).
4. A. Barkalov and L. Titarenko, Logic Synthesis for Compositional Microprogram Control Units, Springer, Berlin (2008).
5. A. A. Barkalov, L. A. Titarenko, and K. N. Efimenko, "Optimization of circuits of compositional microprogram control units implemented on FPGA," Cybernetics and Systems Analysis, **47**, No. 1, 166–174 (2011).
6. R. I. Grushnitskii, A. Kh. Mursaev, and E. P. Ugryumov, Design of Systems on the Basis of PLDs [in Russian], BHV, St. Petersburg (2002).
7. V. V. Solov'ev and A. S. Klimovich, Logic Design of Digital Systems on the Basis of Programmable Logic Integrated Circuits [in Russian], Goryachaya Liniya-Telekom, Moscow (2008).
8. M. Kuzelin, Xilinx EPLDs: The Spartan-3 FPGA Family, http://chip-news.ru/archive/chipnews/200305/2.html
9. R. Senhadji-Navarro, I. Garcia-Vargas, G. Jiménez-Moreno, and A. Civit-Ballcels, "ROM-based FSM implementation using input multiplexing in FPGA devices," Electronics Letters, **40**, No. 20, 1249–1251 (2004.)
10. M. Rawski, H. Selvaraj, and T. Łuba, "An application of functional decomposition in ROM-based FSM implementation in FPGA devices," J. of Syst. Archit., **51**, Nos. 6–7, 424–434 (2005).
11. V. Sklyarov, "Synthesis and implementation of RAM-based finite state machines in FPGAs," in: Proc. 10th Intern. Conf. "Field-programmable logic and applications: The roadmap to reconfigurable computing (FPL 2000)," Villach, Austria (2000), pp. 718–727.
12. A. Tiwari and K. A. Tomko, "Saving power by mapping finite-state machines into embedded memory blocks in FPGAs," in: Proc. Conf. on Design, Automation, and Test in Europe (DATE '04), Vol. 2, Paris (2004), pp. 916–921.
13. E. Garcia, "Creating finite state machines using true dual-port fully synchronous selectRAM blocks," Xcell J., No. 38, 36–38 (2000).
14. A. A. Barkalov, L. A. Titarenko, and S. A. Tsololo, "Optimization of a logic circuit implementing a Moore automaton in CPLD basis," Cybernetics and Systems Analysis, **45**, No. 5, 835–841 (2009).
15. A. Barkalov and L. Titarenko, Logic Synthesis for FSM-Based Control Units, Springer, Berlin (2009).
16. B. S. Frenkel and M. S. Kuzmich, Xilinx WebPACK ISE, http://ru.wikibooks.org/wiki/ Xilinx_WebPACK_ISE.
17. Altera Design and Programming Tools, http://www.altera.ru/cgi-bin/go?19.