Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 4, 751–761
DOI: 10.2478/v10006-010-0057-8

amcs

REDUCTION IN THE NUMBER OF LUT ELEMENTS FOR CONTROL UNITS WITH CODE SHARING

ALEXANDER BARKALOV, LARYSA TITARENKO, JACEK BIEGANOWSKI

Institute of Computer Engineering and Electronics
University of Zielona Góra, Podgórna 50, 65–246 Zielona Góra, Poland
e-mail: {A.Barkalov,L.Titarenko,J.Bieganowski}@iie.uz.zgora.pl

Two methods are proposed targeted at reduction in the number of look-up table elements in logic circuits of compositional microprogram control units (CMCUs) with code sharing. The methods assume the application of field-programmable gate arrays for the implementation of the combinational part of the CMCU, whereas embedded-memory blocks are used for implementation of its control memory. Both methods are based on the existence of classes of pseudoequivalent operational linear chains in a microprogram to be implemented. Conditions for the application of the proposed methods and examples of design are shown. Results of conducted experiments are given.

Keywords: compositional microprogram control unit, code sharing, operational linear chain, field-programmable gate arrays, look-up table, design, embedded-memory block.

1. Introduction

Very often a system includes a control unit (CU) to coordinate the interplay of all system blocks (Navabi, 2007). A particular model chosen to represent a control unit depends strongly on peculiarities of the microprogram to be implemented (Barkalov and Titarenko, 2008).

If the number of control algorithm operator vertices is at least twice less than that of operational linear chains (OLCs), then the model of the compositional microprogram control unit (CMCU) can be used to interpret this control algorithm (Adamski and Barkalov, 2006). Let us point out that the model of the CMCU can be used in any digital system, not only in computers. Field-programmable gate arrays (FPGAs) are widely used for the implementation of control units (Navabi, 2007; Maxfield, 2004). As a rule, these devices include a lot of look-up table (LUT) elements with a very limited number of inputs (Altera, 2010; Xilinx, 2010), as well as configurable embedded memory blocks (EMBs).

The problem of the reduction in the hardware amount in the logic circuit of a control unit is still a task of great importance (Maxfield, 2004; Kam *et al*., 1998; Kania, 2004; Solovjev and Klimowicz, 2008). Its solution allows decreasing such characteristics as the cost of the circuit, the number of chips, power consumption and so on (Micheli, 1994). In the case of the FPGA, this problem can be solved by decreasing the number of input variables in each of the functions to be implemented (Barkalov and Titarenko, 2008). The limited number of inputs per LUT (up to six) results in the necessity of functional decomposition of implemented functions (Scholl, 2001). In turn, this results in a slow-down of the control unit because of the increase in the number of levels in its combinational part. The number of levels can be decreased due to the use of EMBs for implementing some parts of a control unit (Borowik *et al*., 2007). This approach is used in the CMCU, too (Barkalov and Titarenko, 2008; Titarenko and Bieganowski, 2009). It is known that in the case of finite-state-machines (FSMs) (Baranov, 2008), the appropriate state assignment (Micheli, 1994; Barkalov *et al*., 2006; Czerwiński and Kania, 2004; Escherman, 1993) is an effective tool for hardware amount optimization. In the case of the CMCU, only its model with code sharing gives such a possibility. In this article we propose two possible solutions to the hardware amount decrease problem for the CMCU with code sharing implemented using FPGA chips based on LUT elements and EMBs.

2. Background of CMCU with code sharing

Let a microprogram to be implemented be represented by a graph-scheme of algorithm (GSA) (Baranov, 2008) with the set of vertices $B = b_0, b_E \cup E_1 \cup E_2$ and the set

of arcs E. Here b_0 is an initial vertex, b_E is a final vertex, E_1 is a set of operator vertices and E_2 is a set of conditional vertices. An operator vertex $b_q \in E_1$ contains a collection of microoperations $Y(b_q) \subseteq Y$, where $Y = \{y_1, \ldots, y_N\}$ is a set of microoperations. A conditional vertex $b_q \in E_2$ contains some element $x_e \in X$, where $X = \{x_1, \ldots, x_L\}$ is a set of logical conditions.

Let the set $C = \{\alpha_1, \ldots, \alpha_G\}$ be formed for a GSA Γ, where $\alpha_g \in C$ is an operational linear chain. An OLC $\alpha_g \in C$ is a sequence of operator vertices such that each pair of its adjacent components corresponds to some arc from the set E. Each OLC $\alpha_g \in C$ has only one output O_g and an arbitrary number of inputs (Adamski and Barkalov, 2006).

Let us name as a linear GSA a GSA Γ where the number of its operational vertices exceeds at least twice the number of its operational linear chains.

Let each vertex $b_g \in E_1$ correspond to the microinstruction MI_q with address $A(b_q)$, and let this address have R bits, where

$$R = \lceil \log_2 M \rceil. \tag{1}$$

Let each OLC $\alpha_g \in C$ include F_g components, and let $|C| = G$. Let $Q = \max(F_1, \ldots, F_G)$. Encode each OLC $\alpha_g \in C$ by the binary code $K(\alpha_g)$ using variables $\tau_r \in \tau$, where $|\tau| = R_1$ and

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

Encode each component of the OLC $\alpha_g \in C$ by the binary code $K(b_q)$ using variables $T_r \in T$, where $|T| = R_2$ and

$$R_2 = \lceil \log_2 Q \rceil. \tag{3}$$

The encoding of components should be executed in such a manner that the condition

$$K(b_{gi}) = K(b_{gi-1}) + 1 \quad (i = \overline{1, F_g}) \tag{4}$$

is met for each OLC $\alpha_g \in C$. If the condition

$$R = R_1 + R_2 \tag{5}$$

holds, then the GSA Γ can be interpreted by the CMCU with code sharing U_1 (Fig. 1).

In the CMCU U_1, the address of the microinstruction corresponding to component b_q of the OLC $\alpha_g \in C$ is represented as

$$A(b_q) = K(\alpha_g)\ \&\ K(b_q), \tag{6}$$

where & is the concatenation operator. The block of input addresses (BIA) generates input memory functions

$$\begin{aligned} \Phi &= \Phi(T, X), \\ \Psi &= \Psi(T, X), \end{aligned} \tag{7}$$

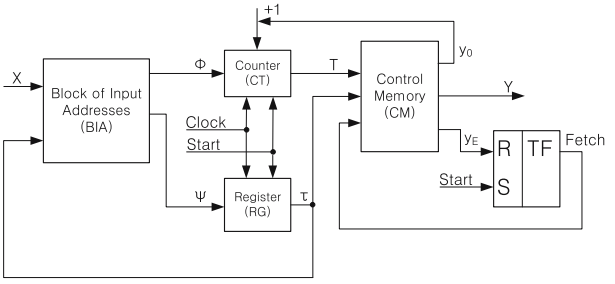

Fig. 1. Structural diagram of the CMCU U_1.

used to load a component code into a counter CT and a code of the OLC into a register RG, respectively. A control memory CM keeps microoperations $y_n \in Y$ and special variables y_0 and y_E. If $y_0 = 1$, then the current content of CT is incremented, otherwise both CT and RG are loaded from the BIA. The first case corresponds to the transition from any component of the OLC α_g except its output. The second case corresponds to the transition from the OLC output. If $y_E = 1$, then a flip-flop TF is cleared, the variable $\mathrm{Fetch} = \emptyset$ and the operation of the CMCU is terminated. It corresponds to the vertex b_E of the GSA. Pulse Start is used to load zero codes into both RG and CT, which corresponds to the address of the first microinstruction. At the same time, the flip-flop TF is set up, $\mathrm{Fetch} = 1$ and microinstructions can be read from CM. Pulse Clock is used for timing the CMCU.

The CMCU U_1 can be viewed as a Moore FSM and chains $\alpha_G \in C$ correspond to internal states of the FSM. The codes $K(\alpha_g)$ do not depend on the codes of components. Therefore, all well-known state assignment methods can be used for the optimization of the BIA circuit. It is shown by Kołopieńczyk (2008) that for linear GSAs the model of the CMCU with code sharing always consumes less amount of LUT elements in comparison with the classical Moore FSM. In the paper, we propose two approaches to the reduction in the number of LUT elements in the logic circuit of the BIA block.

3. Proposed approaches

Our approaches are based on the existence of a pseudoequivalent OLC in the GSA Γ. Recall that the OLC $\alpha_i, \alpha_j \in C$ are pseudoequivalent OLCs if their outputs are connected with the input of the same vertex (Adamski and Barkalov, 2006). Let $\Pi_C = \{B_1, \ldots, B_I\}$ be the partition of the set C by the classes of the pseudoequivalent OLC. Let us construct a set $\Pi_0 \subseteq \Pi_C$, where $B_i \in \Pi_0$ if the outputs of the OLC $\alpha_g \in B_i$ are not connected with the final vertex b_E. Encode each class $B_i \in \Pi_0$ by a binary code $K(B_i)$ with R_3 bits, where

$$R_3 = \lceil \log_2 |\Pi_0| \rceil. \tag{8}$$

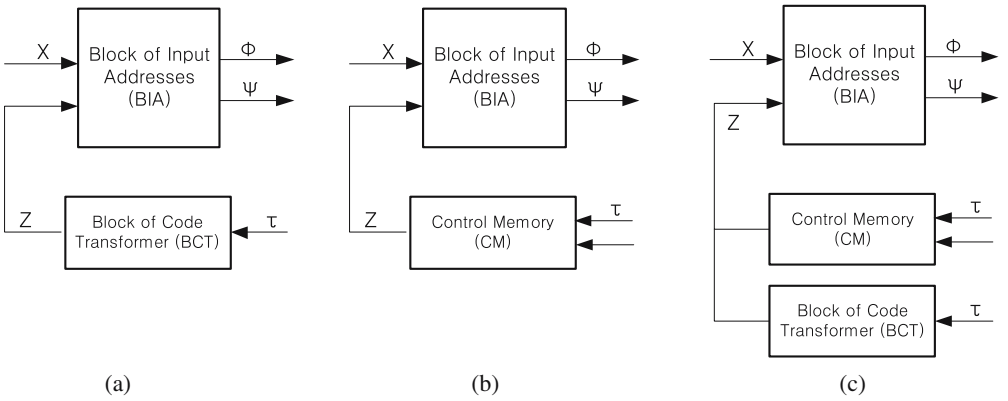

Fig. 2. Generation of $K(B_i)$ codes: using the block of code transformer (a), using control memory (b), using control memory and the block of code transformer (c).

Let variables $z_r \in Z$ be used for such encoding, where $|Z| = R_3$. Let the condition

$$2^{R_2} > F_g \tag{9}$$

hold for all OLCs $\alpha_g \in C_1$, where $C_1 \subseteq C$ is a set of OLC from classes $B_i \in \Pi_0$.

It was shown by Kołopieńczyk (2008) that the number of LUTs needed to implement the BIA can be reduced by about 50% when codes $K(B_i)$ are used to make transitions instead of OLC addresses. The results of Kołopieńczyk (2008) were compared with the base structure of a compositional control unit with code sharing. The method presented by Kołopieńczyk (2008) generates codes $K(B_i)$ using a combinational circuit called the block of code transformer (BCT) (Fig. 2(a)). In the paper, we propose to store codes $K(B_i)$ in control memory to partially (Fig. 2(c)) or completely remove the BCT (Fig. 2(b)).

In our methods we utilize some free areas in control memory that are usually left unused, because the organization memory block is limited to some fixed variants. For example, in the Xilinx II Pro family, the embedded memory block can be organized as 16K $\times$ 1 bit, 8K $\times$ 2 bits, 4K $\times$ 4 bits, 2K $\times$ 9 bits, 1K $\times$ 18 bits, 512 $\times$ 36 bits (Xilinx, 2010). There are two possible variants of putting codes $K(B_i)$ into control memory: by adding some control microinstructions (microprogram expansion) or by adding a field to each microinstruction (microinstruction extension). Let us name a circuit with microprogram expansion as U_2 and circuit with microinstruction extension as U_3. The microinstruction formats used in U_2 and U_3 are shown in Fig. 3.

The most significant bit of each format contains the value of the variable y_0. If $y_0 = 1$ (Fig. 3(a)), then the microinstruction contains the field FY with the code of collection of microoperations to be executed. If $y_0 = 0$ (Fig. 3(b)), then the microinstruction contains the field

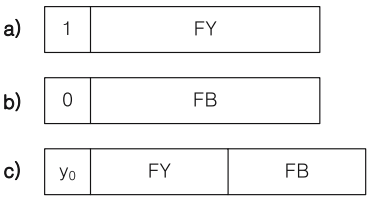

Fig. 3. Microinstruction formats.

FB with the code $K(B_i)$. Let us name the first of them as an operational microinstruction (OMI) and the second as a control microinstruction (CMI). Let us insert an additional component MC_g in each OLC $\alpha_g \in C_1$, and let this component correspond to a CMI with the code $K(B_i)$, where $\alpha_g \in B_i$.

In the CMCU U_2, the block BIA implements the systems

$$\Phi = \Phi(Z, X), \tag{10}$$

$$\Psi = \Psi(Z, X), \tag{11}$$

while the other elements of U_2 have the same meaning as their counterparts for U_1. The functions (10)–(11) are generated when the concatenation of the contents of RG and CT represents an address of the control microinstruction. In this case, the data-path of the controlled digital system is in the idle state. This can be achieved if the synchronization of a data-path is controlled by variable y_0. In this article, we propose a synthesis method for U_2 which includes the following steps:

1. Construction of sets C, C_1, Π_C and Π_0 for a graph-scheme of the algorithm Γ.
2. Including an additional component into each OLC $\alpha_g \in C_1$.
3. Encoding of the OLC $\alpha \in C$, OLC components and classes $B_i \in \Pi_0$.

4. Construction of the control memory content.
5. Construction of the transition table of the CMCU U_2.
6. Implementation of the CMCU logic circuit.

This approach can be applied only if Condition (9) is satisfied. Otherwise, it leads to an increase in the value of R_2, the violation of (5), and code sharing makes no sense (Adamski and Barkalov, 2006). If Condition (9) is violated, we propose to use the CMCU U_3 with the extended microinstruction format shown in Fig. 3(c).

If $y_0 = 1$, then it corresponds to OMI where the content of field FB is ignored. If $y_0 = 0$, then it corresponds to the output of a particular OLC $\alpha_g \in C_1$. In this case, a system data-path executes microoperations represented by the field FY, and CMCU transition depends on the code $K(B_i)$ from the field FB.

In the CMCU U_3, all elements have the same meaning as their counterparts for U_2. The only difference between U_2 and U_3 is in the organization of their control memory blocks. We should point out that the number of inputs for LUT elements of the BIA is decreased if $R_3 < R_1$. This can result in a decrease in the numbers of LUT elements and their levels in the circuit of the BIA in comparison with the CMCU U_1.

Our analysis of CMCU U_2 and U_3 shows that the latter requires more bits in an output word of its control memory. In the case of one-hot encoding of microoperations (Adamski and Barkalov, 2006), the CMCU U_2 requires control memory with

$$n_1 = \max(2 + N, 2 + R_3) \tag{12}$$

bits. At the same time, this value for U_3 is determined as

$$n_2 = 2 + N + R_3. \tag{13}$$

The number 2 in both (12) and (13) is added to take into account the bits for keeping the additional bits y_0 and y_E.

Let us point out that such components of the CMCU as the BIA, CT, RG and TF are implemented using LUT elements, whereas control memory is implemented using EMBs of the same FPGA chip. These blocks have a fixed number of outputs denoted here as t. In reality, $t = 1, 2, 4, 8, 16$ (Maxfield, 2004). This means that

$$R_4 = \left\lceil \frac{2+N}{t} \right\rceil t - 2 - N \tag{14}$$

bits are free and can be used to represent the code $K(B_i)$. If the condition

$$R_3 \leq R_4 \tag{15}$$

is violated, we can use a code transformer to implement $(R_3 - R_4)$ bits of the code $K(B_i)$. This leads to the CMCU U_4 (Fig. 2(c)), where the block of code transformer implements some bits of the code $K(B_i)$.

In the case of the CMCU U_4, the BIA implements the systems

$$\Phi = \Phi(Z, V, X), \tag{16}$$

$$\Psi = \Psi(Z, V, X), \tag{17}$$

and the BCT implements the system

$$V = V(\tau). \tag{18}$$

The other components of U_4 have the same meaning as their counterparts for U_3. Let us point out that variables $z_r \in Z$ represent R_4 leftmost bits of the code $K(B_i)$, whereas variables $v_r \in V$ represent the remaining

$$R_5 = R_3 - R_4 \tag{19}$$

bits. Obviously, the CMCU U_4 can be reduced to U_3 if the condition (15) is true. In this article, we propose a synthesis method for U_4 which includes the following steps:

1. Construction of sets C, C_1, Π_C and Π_0 for a graph-scheme of the algorithm Γ.
2. Encoding of the OLC $\alpha_g \in C$, their components and classes $B_i \in \Pi_0$.
3. Construction of the control memory content.
4. Construction of a transition table of the CMCU U_4.
5. Construction of the table of the BCT.
6. Implementation of the CMCU logic circuit.

4. Examples of the application of the proposed methods

Let a microprogram to be implemented be represented by a GSA Γ_1 (Fig. 4). Applying the approaches of Barkalov and Titarenko (2008), the following sets can be found for the GSA Γ_1: $C = \{\alpha_1, \ldots, \alpha_8\}$, $C_1 = \{\alpha_1, \ldots, \alpha_7\}$, $\Pi_C = \{B_1, \ldots, B_4\}$, $\Pi_0 = \{B_1, B_2, B_3\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3, \alpha_4\}$, $B_3 = \{\alpha_5, \alpha_6, \alpha_7\}$, $B_4 = \{\alpha_8, \}$ $\alpha_1 = \langle b_1, b_2\rangle$, $\alpha_2 = \langle b_3, b_4, b_5\rangle$, $\alpha_3 = \langle b_6, b_7\rangle$, $\alpha_4 = \langle b_8, b_9, b_{10}\rangle$, $\alpha_5 = \langle b_{11}, b_{12}\rangle$, $\alpha_6 = \langle b_{13}, b_{14}\rangle$, $\alpha_7 = \langle b_{15}, b_{16}, b_{17}\rangle$, $\alpha_8 = \langle b_{18}, \ldots, b_{21}\rangle$. This means that $Q = 4$, $R_2 = 2$, $T = \{T_1, T_2\}$, $G = 8$, $R_1 = 3$, $\tau = \{\tau_1, \tau_2, \tau_3\}$, $\Psi = \{D_1, D_2, D_3\}$, $\Phi = \{D_4, D_5\}$. Because $M = 21$, $R = 5$, the condition (5) holds and code sharing makes sense. Because $\alpha_8 \ni C_1$, the condition (9) is satisfied and all the OLC $\alpha_g \in C_1$ can be modified. After the modification, we have the OLC $\alpha_1 = \langle b_1, b_2, MC_1\rangle, \ldots, \alpha_7 = \langle b_{15}, b_{16}, b_{17}, MC_7\rangle$.

Let us encode OLC $\alpha_g \in C$ in an arbitrary manner, namely, $K(\alpha_1) = 000, \ldots, K(\alpha_8) = 111$. Let the code 00 be assigned to the first component of any OLC $\alpha_g \in C$, the code 01 to the second, the code 10 to the third, and the

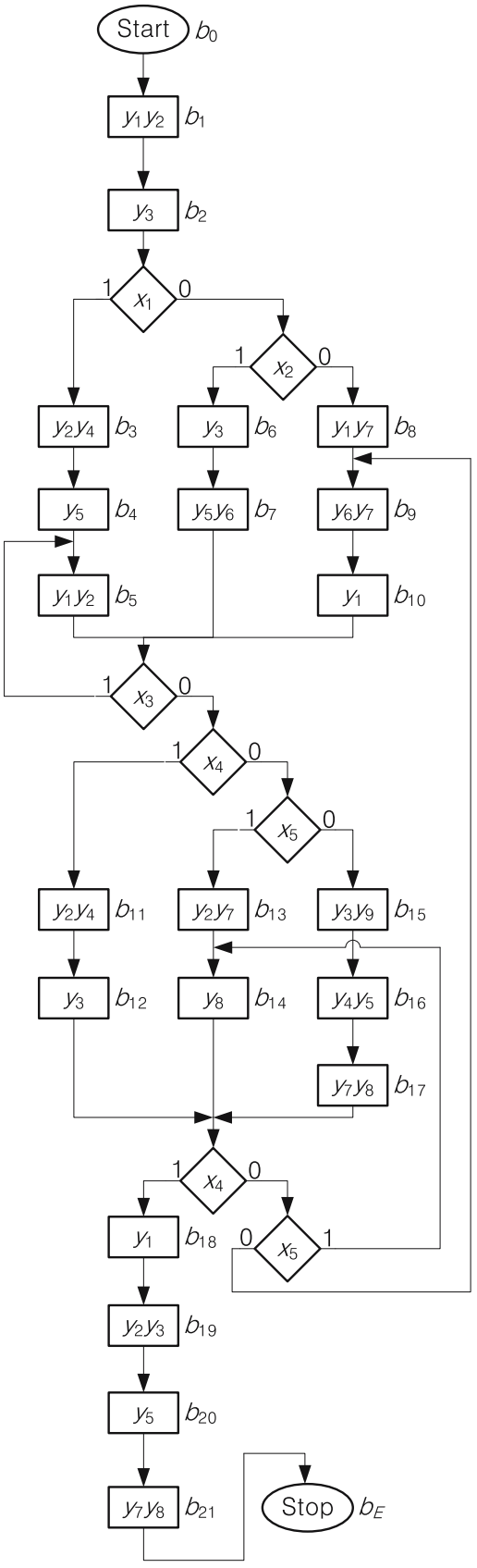

Fig. 4. Initial graph-scheme of Algorithm Γ_1.

code 11 to the fourth. Let us encode classes $B_i \in \Pi_0$ in an arbitrary manner, namely, $K(B_1) = 00$, $K(B_1) = 01$, $K(B_3) = 10$. Now microinstruction addresses are shown in Fig. 5, where microinstructions are represented by the corresponding vertices and additional components.

T_1T_2 \ $\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110	111
00	b_1	b_3	b_6	b_8	b_{11}	b_{13}	b_{15}	b_{18}
01	b_2	b_4	b_7	b_9	b_{12}	b_{14}	b_{16}	b_{19}
10	MC_1	b_5	MC_3	b_{10}	MC_5	MC_6	b_{17}	b_{20}
11	–	MC_2	–	MC_4	–	–	MC_7	b_{21}

Fig. 5. Microinstruction addresses for the CMCU $U_2(\Gamma_1)$.

Let $U_i(\Gamma_j)$ mean that CMCU U_i is used to interpret a GSA Γ_j. From Fig. 5 in can be inferred, for example, that $A(b_{10}) = K(\alpha_4) \cdot K(b_{10}) = 01110$; $A(MC_6) = K(\alpha_6) \cdot K(MC_6) = 10110$ and so on.

To get the control memory content, we should replace each vertex $b_q \in B_1$ by the set $Y(b_q) \cup \{y_0\}$. Each component MC_g is replaced by the code $K(B_i)$, where $\alpha_g \in B_i$. The vertex b_{21} is replaced by the set $Y(b_{21}) \cup \{y_E\}$.

For example, the vertex b_8 contains the set $Y(b_8) = \{y_1, y_7\}$. This means that the memory cell with the address $A(b_8) = 01100$ contains a binary code corresponding to y_0, y_1, y_7. Next, the vertex b_{21} is connected with the input of the final vertex b_E, and $Y(b_{21}) = \{y_7, y_8\}$, $A(b_{21}) = 11111$. Then, the memory cell with this address includes a binary code corresponding to y_7, y_8, y_E. Finally, the memory cell with the address 01011 corresponds to MC_4. Because this control microinstruction follows the vertex b_{10}, which is the output of OLC $\alpha \in B_2$, this memory cell includes the code $K(B_2) = 01$. The contents of all other cells can be found in the same manner.

The transitions from the outputs of the OLC $\alpha_g \in C_1$ can be represented by the following system of transition formulae (Baranov, 2008):

$$\begin{aligned} b_2 &\rightarrow x_1b_3 \vee \bar{x}_1x_2b_6 \vee \bar{x}_1\bar{x}_2b_8, \\ b_5, b_7, b_{10} &\rightarrow x_3b_5 \vee \bar{x}_3x_4b_{11} \vee \\ &\quad \bar{x}_3\bar{x}_4x_5b_{13} \vee \bar{x}_3\bar{x}_4\bar{x}_5b_{15}, \\ b_{12}, b_{14}, b_{17} &\rightarrow x_4b_{18} \vee \bar{x}_4x_5b_{14} \vee \bar{x}_4\bar{x}_5b_9. \end{aligned} \tag{20}$$

As follows from (20), the outputs of the pseudoequivalent OLC $\alpha_g \in B_i$ are described by one line of the system. This allows replacing the outputs of OLC $\alpha_g \in B_i$ by the corresponding class $B_i \in \Pi_c$. This leads

to the system of generalized transition formulae:

$$\begin{aligned} B_1 &\rightarrow x_1 b_3 \vee \bar{x}_1 x_2 b_6 \vee \bar{x}_1 \bar{x}_2 b_8, \\ B_2 &\rightarrow x_3 b_5 \vee \bar{x}_3 x_4 b_{11} \vee \bar{x}_3 \bar{x}_4 x_5 b_{13} \vee \\ &\quad \bar{x}_3 \bar{x}_4 \bar{x}_5 b_{15}, \\ B_3 &\rightarrow x_4 b_{18} \vee \bar{x_4} x_5 b_{14} \vee \bar{x}_4 \bar{x_5} b_9. \end{aligned} \tag{21}$$

The system of transition formulae can be transformed into a transition table of the CMCU with the following columns: B_i, $K(B_i)$, b_q, $A(b_q)$, X_h, Ψ_h, Φ_h, h. Here $\Psi_h(\Phi_h)$ is the collection of input memory functions that are equal to 1 for the h-th transition of the CMCU ($h = 1, \ldots, H$). The transition number h is determined by a conjunction of some logical conditions X_h ($h = 1, \ldots, H$). In our example, $H = 10$ and the table of CMCU transitions is represented by Table 1.

Table 1. Transition table of the CMCU $U_2(\Gamma_1)$.

B_i	$K(B_i)$ $z_1 z_2$	b_q	$A(b_q)$ $\tau_1\tau_2\tau_3 T_1 T_2$	X_h	Ψ_h	Φ_h	h
B_1	00	b_3	00100	x_1	D_3	–	1
		b_6	01000	$\bar{x}_1 x_2$	D_2	–	2
		b_8	01100	$\bar{x}_1 \bar{x}_2$	$D_2 D_3$	–	3
B_2	01	b_5	00110	x_3	D_3	D_4	4
		b_{11}	10000	$\bar{x}_3 x_4$	D_1	–	5
		b_{13}	10100	$\bar{x}_3 \bar{x}_4 x_5$	$D_1 D_3$	–	6
		b_{15}	11000	$\bar{x}_3 \bar{x}_4 \bar{x}_5$	$D_1 D_2$	–	7
B_3	10	b_{18}	11100	x_4	$D_1 D_2 D_3$	–	8
		b_{14}	10101	$\bar{x}_4 x_5$	$D_1 D_3$	D_5	9
		b_9	01101	$\bar{x}_4 \bar{x}_5$	$D_2 D_3$	D_5	10

This table is used to derive the systems (10) and (11). The following equations can be derived, for example, from Table 1:

$$\begin{aligned} D_1 &= \bar{z}_1 z_2 \bar{x}_3 \vee z_1 \bar{z}_2 x_4 \vee z_1 \bar{z}_2 \bar{x}_4 x_5, \\ D_4 &= \bar{z}_1 z_2 x_3, \\ D_5 &= z_1 \bar{z}_2 \bar{x}_4. \end{aligned}$$

The first term in function D_1 corresponds to the lines 5–7 of Table 1, and the only term in function D_5 corresponds to the lines 9 and 10 of Table 1. The implementation of the CMCU U_2 logic circuit is reduced to that of the systems (10)–(11) using LUT elements and the implementation of its control memory using EMBs of FPGA chips. Some industrial or academic packages such as SIS, Xilinx ISE or Altera Quartus II (Altera, 2010; Xilinx, 2010; Sentovich *et al.*, 1992) can be used to solve this problem.

Let us discuss a bit different case, when a microprogram is represented by a GSA Γ_2 (Fig. 6). Applying the approaches by Barkalov and Titarenko (2008), the following sets can be found for the GSA Γ_2: $C = \{\alpha_1, \ldots, \alpha_7\}$, $\Pi_C = \{B_1, \ldots, B_4\}$, $C_1 = \{\alpha_1, \ldots, \alpha_6\}$, $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3\}$, $B_3 = \{\alpha_4, \alpha_5, \alpha_6\}$, $B_4 = \{\alpha_7, \}$, where $\alpha_1 = \langle b_1, b_2, b_3\rangle$, $\alpha_2 = \langle b_4, b_5, b_6, b_7\rangle$, $\alpha_3 = \langle b_8, b_9\rangle$, $\alpha_4 = \langle b_{10}, b_{11}, b_{12}, b_{13}\rangle$, $\alpha_5 = \langle b_{14}, b_{15}, b_{16}\rangle$, $\alpha_6 = \langle b_{17}, b_{18}, b_{19}, b_{20}\rangle$, $\alpha_7 = \langle b_{21}, b_{22}, b_{23}\rangle$. This means that $M = 23$, $R = 5$, $G = 7$, $R_1 = 3$, $Q = 4$, $R_2 = 2$ and $T = \{T_1, T_2\}$, $\tau = \{\tau_1, \tau_2, \tau_3\}$, $\Psi = \{D_1, D_2, D_3\}$, $\Phi = \{D_4, D_5\}$. As we can see, the GSA Γ_2 includes $N = 13$ different microoperations.

The analysis of the GSA Γ_2 shows that $R_1 + R_2 = R$, thus the application of code sharing makes sense. Because the condition (9) is violated for the OLC $\alpha_2, \alpha_4, \alpha_6 \in C_1$, the model U_3 should be applied. Let $q = 3$, $t = 4$, where t is the number of PROM outputs. It can be found that $R_3 = 2$, $n_2 = 17$, $R_4 = 1$ and the condition (15) is violated. This means that the model $U_4(\Gamma_2)$ should be used, where $V = \{v_1\}$, $Z = \{z_1\}$. Let us discuss this design example.

Let $K(\alpha_1) = 000, \ldots, K(\alpha_7) = 110$, and let the code 00 be assigned to the first component of any OLC $\alpha_g \in C, \ldots$, the code 11 to the fourth component of any OLC $\alpha_g \in C$. Now microinstruction addresses for the CMCU $U_4(\Gamma_2)$ are shown in Fig. 7.

Let us encode the classes $B_i \in \Pi_0$ in a trivial way (Table 2).

Table 2. Codes of the classes $B_i \in \Pi_0$.

B_i	$K(B_i)$	
	v_1	z_q
B_1	0	0
B_2	0	1
B_2	1	0

As follows from the analysis of the GSA Γ_2, $Y(b_4) = \{y_2, y_3\}$, $Y(b_5) = \{y_6\}$, $Y(b_6) = \{y_3, y_5, y_7\}$, $Y(b_7) = \{y_3, y_6\}$. The part of the control memory content for the OLC $\alpha_2 \in B_2$ is shown in Table 3.

Table 3. Part of the control memory content.

Address $\tau_1\tau_2\tau_3 T_1 T_2$	Content	Reference
00100	$y_0 y_2 y_3$	b_4
00101	$y_0 y_6$	b_5
00110	$y_0 y_3 y_5 y_7$	b_6
00111	$y_3 y_6 z_1$	b_7

Table 3 shows the main principles of control memory content construction, which are as follows. If a vertex $b_q \in E_1$ is not the output of an OLC $\alpha_g \in C_1$, then the memory cell with the address $A(b_q)$ contains $Y(b_q)$ and y_0. Otherwise, this cell contains $Y(b_q)$ and variables

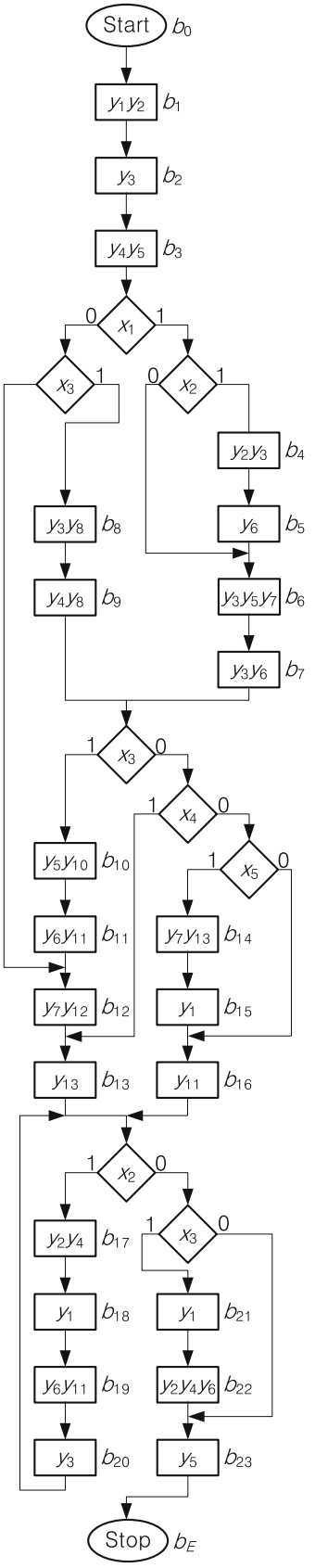

Fig. 6. Initial graph-scheme of Algorithm Γ_2.

T_1T_2 \ $\tau_1\tau_2\tau_3$	000	001	010	011	100	101	110
00	b_1	b_4	b_8	b_{10}	b_{14}	b_{17}	b_{21}
01	b_2	b_5	b_9	b_{11}	b_{15}	b_{18}	b_{22}
10	b_3	b_6	–	b_{12}	b_{16}	b_{19}	b_{23}
11	–	b_7	–	b_{13}	–	b_{20}	–

Fig. 7. Microinstruction addresses for the CMCU $U_4(\Gamma_2)$.

$z_r \in Z$, which are equal to 1 in the code $K(B_i)$, where $\alpha_g \in B_i$. If b_q is the output of OLC $\alpha_g \ni C_1$, then the corresponding memory cell contains $Y(b_q)$ and y_E.

Using the same rules as in the case of the CMCU $U_2(\Gamma_1)$, the following system of the generalized transition formulae can be constructed for the GSA Γ_2:

$$\begin{aligned} B_1 &\rightarrow x_1x_2b_4 \vee x_1\bar{x}_2b_6 \vee \bar{x}_1x_3b_8 \vee \bar{x}_1\bar{x}_3b_{12}, \\ B_2 &\rightarrow x_3b_{10} \vee \bar{x}_3x_4b_{13} \vee \bar{x}_3\bar{x}_4x_5b_{14} \vee \\ &\quad \bar{x}_3\bar{x}_4\bar{x}_5b_{16}, \\ B_3 &\rightarrow x_2b_{17} \vee \bar{x}_2x_3b_{21} \vee \bar{x}_2\bar{x}_3b_{23}. \end{aligned} \tag{22}$$

This system is used to construct the transition table of the CMCU $U_4(\Gamma_2)$ with $H_4(\Gamma_2) = 11$ lines (Table 4).

Table 4. Transition table of the CMCU $U_4(\Gamma_2)$.

B_i	$K(B_i)$ v_1z_1	b_q	$A(b_q)$ $\tau_1\tau_2\tau_3T_1T_2$	X_h	Ψ_h	Φ_h	h
B_1	00	b_4	00100	$x_1\bar{x}_2$	D_3	–	1
		b_6	00110	x_1x_2	D_3	D_4	2
		b_8	01000	$\bar{x}_1x_3$	D_2	–	3
		b_{12}	01110	$\bar{x}_1\bar{x}_3$	D_2D_3	D_4	4
B_2	01	b_{10}	01100	x_3	D_2D_3	–	5
		b_{13}	01111	$\bar{x}_3x_4$	D_2D_3	D_4D_5	6
		b_{14}	10000	$\bar{x}_3\bar{x}_4x_5$	D_1	–	7
		b_{16}	10010	$\bar{x}_3\bar{x}_4\bar{x}_5$	D_1	D_4	8
B_3	10	b_{17}	10100	x_2	D_1D_3	–	9
		b_{21}	11000	$\bar{x}_2x_3$	D_1D_2	–	10
		b_{23}	11010	$\bar{x}_2\bar{x}_3$	D_1D_2	D_4	11

This table is used to derive the systems (16) and (17). After minimization, the following equations can be found, for example, from Table 4:

$$\begin{aligned} D_1 &= \bar{v}_1z_1\bar{x}_3\bar{x}_4 \vee v_1\bar{z}_1, \\ D_2 &= \bar{v}_1\bar{z}_1\bar{x}_1 \vee \bar{v}_1z_1x_3 \vee \\ &\quad \bar{v}_1z_1\bar{x}_3x_4 \vee v_1\bar{z}_1\bar{x}_2, \\ D_5 &= \bar{v}_1z_1\bar{x}_3x_4. \end{aligned}$$

The table of the BCT includes the following columns: α_g, $K(\alpha_g)$, B_i, $K(B_i)$, V_g, g. In our example, this table has $G_4(\Gamma_2) = 3$ lines (Table 5).

Table 5. Table of the BCT CMCU $U_4(\Gamma_2)$.

α_g	$K(\alpha_g)$	B_i	$K(B_i)$	V_g	g
α_4	011	B_3	10	v_1	1
α_5	100				2
α_6	101				3

In the ordinary case, $G_4(\Gamma_i)$ is equal to the number of the OLC $\alpha_g \in B_i$, where $K(B_i)$ includes $v_r \neq 0$ $(r = 1, \ldots, R_5)$. This table is used to derive the system (18). After minimization, we can get the following equation from Table 5:

$$v_1 = \bar{\tau}_1\tau_2\tau_3 \vee \tau_1\bar{\tau}_2. \tag{23}$$

The implementation of the CMCU U_4 logic circuit is reduced to that of the systems (16)–(18) using LUT elements and that of control memory using EMBs of FPGA chips. Some industrial or academic packages (Altera, 2010; Xilinx, 2010; Sentovich *et al.*, 1992) can be used to solve this problem.

5. Analysis of the proposed methods

The proposed methods were tested using Xilinx ISE 12.1. The synthesis was made for Xilinx Virtex 5 FPGA (xc5vlx30-3ff324) and Xilinx XC9500 CPLD (xc9536-5PC44) with area optimization (`opt_mode area level 1`). The outcomes of our research are shown in Table 6. The number of slices (column S_F) and the time of cycle (column T_C) were taken from synthesis reports of Xilinx ISE. We decided to measure results in slices because the methods presented in Table 6 use LUTs and flip-flops. We assume that a slice is utilized when LUT or the flip-flop in that slice is utilized. The circuit U_5 with the Moore FSM was created in the VHDL according to guidelines of Xilinx (2006). The control memory for all CMCU models in Table 6 was implemented using one EMB.

All tested models were generated using original software developed by us. The input file to our application is the KISS file format (Yang, 1991) and the output is an FSM or a CMCU model in the VHDL. The test files (linear GSAs) were taken from Kołopieńczyk (2008).

When we compare results of CMCU models, we can see that the proposed methods give on average about 50% reduction in the area (using the same number of flip-flops and embedded memory blocks) in comparison with the base model. The results are similar to those presented by Kołopieńczyk (2008). When compared with the Moore FSM, we can see that the area can be reduced on the average by about 70% for the FSM with "compact" state encoding. We should recall here that these results were obtained for synthetic linear GSAs and can vary in real circuits.

It is very interesting that these methods can be applied in the case of complex programmable logic devices (CPLDs). In this case, the logic circuit of the BIA is implemented using programmable array logic (PAL) macrocells and control memory can be implemented using some PROM/RAM chips external to a CPLD chip. This is connected with the fact that the use of the proposed method permits decreasing the number of transition table lines from $H_1(U_1)$ to $H_2(U_2)$, where

$$H_1 = \sum_{i=1}^{I_0} E_i \cdot |B_i|, \tag{24}$$

$$H_2 = \sum_{i=1}^{I_0} E_i. \tag{25}$$

In (24)–(25), E_i is the number of transitions from the output of any OLC $\alpha_g \in B_i$, where $B_i \in \Pi_0$. For GSA Γ_1, we have $H_1 = 24$ and $H_2 = 10$.

To estimate the hardware amount of the BIA, let us use the following symbols: q is the number of terms in a PAL macrocell, $E_i(D_r)$ is the number of the terms in function $D_r \in \Phi \cup \Psi$ for CMCU U_i, $Q_i(D_r, q)$ is the number of macrocells required for the implementation of function $D_r \in \Phi \cup \Psi$ of CMCU U_i using PAL macrocells with q terms:

$$Q_i(D_r, q) = \left\lceil \frac{E_i(D_r) - q}{q - 1} \right\rceil + 1. \tag{26}$$

The formula (26) is based on results presented by Kania (2004).

From Table 1 we obtain that: $E_2(D_1) = E_2(D_2) = 3$, $E_2(D_3) = 5$, $E_2(D_4) = E_2(D_5) = 1$. Let $q = 3$. Then $Q_2(D_r, 3) = 1$ for $\Gamma = 1, 2, 4, 5$ and $Q_2(D_3, 3) = 2$. Thus, in the case of the CMCU $U_2(\Gamma_1)$, the logic circuit for the BIA block needs $Q_2(\Gamma_1) = 6$ macrocells and has two levels.

If we construct the transition table for the CMCU $U_1(\Gamma_1)$, we will find that $E_1(D_1) = 9$, $E_1(D_2) = 11$, $E_1(D_3) = 13$, $E_1(D_4) = E_1(D_5) = 3$. Thus, $Q_1(D_1, 3) = 4$, $Q_1(D_2, 3) = 5$, $Q_1(D_3, 3) = 6$, $Q_1(D_4, 3) = Q_1(D_5, 3) = 1$ and $Q_1(\Gamma_1) = 17$ macrocells. Let us point out that this circuit has two levels, too. The following conclusion can be drawn: transition from $U_1(\Gamma_1)$ to $U_2(\Gamma_1)$ allows decreasing the hardware amount in 2.8 times. Of course, more cycles are needed to execute the algorithm Γ_1 using U_2 than in the case of U_1. In this article, we do not estimate this delay in both cases of FPGA and CPLD implementation. To find the value of

Table 6. Results of experiments.

File	L	N	M	G	Q	FPGA									CPLD				
						U_1		U_2		U_3		U_4^*	U_5		U_1	U_2	U_3	U_4^*	U_5
						S_F	T_C	S_F	T_C	S_F	T_C	S_F	S_F	T_C	S_C	S_C	S_C	S_C	S_C
mk01	10	7	86	14	15	37	2.419	19	3.203	15	3.058	2	103	2.743	212	179	120	35	727
mk02	10	7	90	15	10	35	2.805	17	3.061	16	3.061	2	122	2.649	256	201	121	65	758
mk03	7	8	49	11	8	18	2.395	13	3.433	11	3.052	2	30	1.964	139	77	84	18	389
mk04	9	8	57	14	10	32	2.419	18	3.105	15	3.065	2	35	2.568	184	139	108	41	494
mk05	10	9	55	13	9	31	2.425	17	3.057	13	3.053	2	35	2.055	245	147	126	36	481
mk06	8	11	59	14	7	26	2.636	16	2.880	13	3.052	2	31	1.704	159	138	127	32	485
mk07	11	8	71	17	8	45	2.467	24	3.880	16	3.053	3	93	2.783	321	232	130	73	594
mk08	7	7	50	12	9	31	2.880	15	3.206	13	3.058	2	28	2.063	154	102	86	23	400
mk09	7	7	57	13	9	29	2.419	15	3.564	13	3.058	2	30	2.688	230	160	104	32	468
mk10	13	11	93	21	7	68	2.522	28	3.337	21	3.492	4	134	2.533	488	280	214	85	897
mk11	8	8	49	13	6	30	3.053	12	3.052	12	2.967	2	28	1.619	189	102	82	24	431
mk12	11	8	72	19	7	50	2.383	22	2.877	18	3.058	4	119	2.644	371	194	155	49	675
mk13	6	7	38	9	6	15	2.416	9	3.053	8	3.053	2	25	1.690	58	75	56	16	237
mk14	8	7	54	15	6	24	2.347	16	3.026	13	2.967	2	27	1.780	195	97	104	39	474
mk15	8	10	69	15	8	23	2.416	11	3.052	16	2.748	2	76	2.334	240	107	114	40	563
mk16	6	10	72	12	11	28	3.012	15	3.549	13	2.963	2	116	3.139	135	109	82	27	675
mk17	9	7	73	15	8	24	2.347	21	3.729	15	3.491	2	82	2.530	205	115	123	31	605
mk18	6	11	52	13	8	26	2.347	15	3.438	10	2.810	2	25	1.692	152	151	96	34	422
mk19	6	7	47	12	6	4	2.347	11	2.813	10	3.053	2	31	2.515	128	87	68	30	359

L: the number of logical conditions, N: the number of microinstructions in microoperation, M: the number of operational vertices, G: the number of OLC chains, Q: the maximal length of the the OLC chain, S_F: the number of slices, S_C: the number of basic elements (BELS), i.e., AND2, AND3, INV, OR2, OR3, XOR2 in the the PLA macrocell, T_C: the time of the cycle in nanoseconds, U_4^*: the maximal size of the BCT in the CMCU U_4 (the BCT implements all bits of $K(B_i)$), U_5: control unit implemented as a Moore FSM with "compact" state encoding

this delay, some complex statistical experiments should be conducted.

Let us estimate the number of PAL macrocells with $q = 3$ in the logic circuit of the CMCU $U_4(\Gamma_2)$ and $U_1(\Gamma_4)$. As follows from Table 4 $E_4(D_1) = 2$, $E_4(D_2) = 4$, $E_4(D_3) = E_4(D_4) = 5$, $E_4(D_5) = 1$. From Table 5 it can be found that $E_4(v_1) = 2$. Using (26), we can determine $Q_4(D_1, 3) = Q_4(D_5, 3) = Q_4(v_1, 3) = 1$ and $Q_4(D_r, 3) = 2$ for $r = 2, 3, 4$. Thus, $Q_4(\Gamma_2) = 9$ macrocells.

If we construct the transition table for the CMCU $U_1(\Gamma_2)$, we will find that $H_1(\Gamma_2) = 24$, $E_1(D_1) = 5$, $E_1(D_2) = 8$, $E_1(D_3) = E_1(D_4) = 9$, $E_1(D_5) = 2$. This means that $Q_1(D_1, 3) = 2$, $Q_1(D_5, 3) = 1$, $Q_1(D_r, 3) = 4$ for $r = 2, 3, 4$. Thus $Q_1(\Gamma_2) = 15$ macrocells. Therefore, the combinational part of the CMCU $U_4(\Gamma_2)$ is implemented using 1.67 times fewer PAL macrocells. Let us point out that cycle times for both CMCU $U_1(\Gamma_2)$ and $U_4(\Gamma_2)$ are the same. Besides, they use the same number of PROM chips with $t = 4$ outputs to implement their control memories.

6. Conclusion

The aim of the proposed methods is to reduce the number of LUT elements in the combinational part of compositional microprogram control units implemented with FPGA. These methods are based on the existence of classes of pseudoequivalent operational linear chains. The encoding of these classes allows a reduction in the number of variables in each of input memory functions. At the same time, these methods allow reducing the number of terms in input memory functions. Therefore the proposed method can be applied in the case of a CPLD, too.

The first approach suggests the introduction of additional control microinstructions. If the condition (9) takes place, each OLC can be transformed in this way. This leads to the optimization of hardware, but it also has a negative effect connected with additional idle cycles in the digital system data-path. The second approach is based on the inclusion of the class code into the microinstruction and it does not affect the performance, but sometimes an additional block of code transformer is needed so the hardware amount could be increased in comparison with the first approach. Also, note that the second approach is more flexible because it does not depend on the condition (9) and can be applied for any GSA.

Our experiments show that the proposed methods always produce logic circuits with fewer LUT elements compared to the use of the known model of the CMCU with code sharing U_1. These methods produce CMCU circuits with less hardware, when they are implemented with a CPLD, too. This shows once more that reduction methods targeted at CPLDs can be applied for the FPGA and

vice versa. This was first mentioned by Kania (2004). Recall that these methods are useful only if a microprogram to be implemented is represented by a linear GSA.

The proposed methods can be applied to a linear GSA with many long chains of operations. Such long chains can be found in many computational tasks, i.e., MD5 (Rivest, 1992) and SHA (Eastlake and Jones, 2001) algorithms. The MD5 algorithm has 64 unconditional steps, and each step depends on the results of previous calculations. This means that there is no possibility to make operations in paralel and the only way to improve the throughput is to use the pipelined data-path (Jarvinen *et al*., 2005). We should note here that control units described in our article execute microinstructions sequentially, but the microoperations for data-path are executed in parallel. One microinstruction can activate an unlimited number of microoperations in parallel. A sequential control unit does not limit the data-path to be sequential.

References

Adamski, M. and Barkalov, A. (2006). *Architectural and Sequential Synthesis of Digital Devices*, University of Zielona Góra Press, Zielona Góra.

Altera (2010). Altera corpotation webpage, `http://www.altera.com`

Baranov, S. (2008). *Logic and System Design of Digital Systems*, TUT Press, Tallinn.

Barkalov, A. and Titarenko, L. (2008). *Logic Synthesis for Compositonal Microprogram Control Units*, Springer, Berlin.

Barkalov, A., Titarenko, L. and Wiśniewski, R. (2006). Synthesis of compositional microprogram control units with sharing codes and address decoder, *Proceedings of the International Conference on Mixed Design of Integrated Circuits and Systems, MIXDES 2006, Gdynia, Poland*, pp. 397–400.

Borowik, G., Falkowski, B. and Łuba, T. (2007). Cost-efficient synthesis for sequetnial circuits implemented using embedded memory blocks of FPGA's, *Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Cracow, Poland*, pp. 99–104.

Czerwiński, R. and Kania, D. (2004). State assignment method for high speed FSM, *Proceedings of the IFAC Workshop on Programmable Devices and Systems, PDS, Cracow, Poland*, pp. 216–221.

Eastlake, D. and Jones, P. (2001). RFC:3174 US secure hash algorithm 1 (SHA1), *Technical report*, Network Working Group, `http://www.faqs.org/rfcs/rfc3174.html`.

Escherman, B. (1993). State assignment for hardwired VLSI control units, *ACM Computing Surveys* **25**(4): 415–436.

Jarvinen, K., Tommiska, M. and Skytta, J. (2005). Hardware implementation analysis of the MD5 hash algorithm, *HICSS'05: Proceedings of the 38th Annual Hawaii Interenational Conference on System Sciences, Waikoloa, Hi, USA*, p. 298.1.

Kam, T., Villa, T., Brayton, R. and Sangiovanni-Vincentelli, A. (1998). *A Synthesis of Finie State Machines: Functional Optimization*, Kluwer Academic Publishers, Boston, MA.

Kania, D. (2004). *Logic Synthesis for PAL-Based Complex Programmable Logic Devices*, Scientific Fascicles of the Silesian University of Technology, Gliwice, (in Polish).

Kołopieńczyk, M. (2008). *Application of Address Converter for Decreasing Memory Size of Compositional Microprogram Control Unit with Code Sharing*, University of Zielona Góra Press, Zielona Góra.

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Academic Press, Orlando, FL.

Micheli, G.D. (1994). *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, NY.

Navabi, Z. (2007). *Embedded Core Design with FPGAs*, McGraw-Hill, New York, NY.

Rivest, R. (1992). RFC:1312 the MD5 message-digest algorithm, *Technical report*, Network Working Group, `http://www.faqs.org/rfcs/rfc1312.html`.

Scholl, C. (2001). *Functional Decomosition with Application of FPGA Synthesis*, Kluwer Academic Publishers, Boston, MA.

Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K. and Sangiovanni-Vincentelli, A.L. (1992). SIS: A system for sequential circuit synthesis, *Technical Report UCB/ERL M92/41*, EECS Department, University of California, Berkeley, CA.

Solovjev, V.V. and Klimowicz, A. (2008). *Logic Design for Digital Systems on the Base of Programmable Logic Integerated Circuits*, Hot Line-Telecom, Moscow, (in Russian).

Titarenko, L. and Bieganowski, J. (2009). Optimization of compositional microprogram control unit by modification of microinstruction format, *Electronics and Telecommunication Quarterly* **55**(2): 201–214.

Xilinx (2006). *Xilinx Synthesis and Simulation Design Guide*, Xilinx, `http://www.xilinx.com/itp/xilinx9/books/docs/sim/sim.pdf`.

Xilinx (2010). Xilinx corpotation webpage, `http://www.xilinx.com`.

Yang, S. (1991). Logic synthesis and optimization benchmarks user guide, *Technical report*, Microelectronic Center of North Carolina, Research Triangle Park, NC 27709-2889.

Alexander Barkalov worked at the Donetsk National Technical University (DNTU), Ukraine, from 1976 till 1996 as a tutor. He cooperated actively with the Kiev Institute of Cybernetics (IC) named after Victor Glushkov. He obtained a doctoral degree (informatics) in 1995 from the IC. From 1996 till 2003 he worked as a professor of the DNTU. Since 2003 he has been working as a professor at the Faculty of Electrical Engineering, Computer Science and Telecommunications of the University of Zielona Góra, Poland.

Larysa Titarenko obtained a doctoral degree (telecommunications) in 2005 from the Kharkov National University of Radioelectronics (KNURE), Ukraine. Till September 2003, she worked as a professor of the KNURE. Since 2005 she has been working as a professor at the Faculty of Electrical Engineering, Computer Science and Telecommunications of the University of Zielona Góra, Poland.

Jacek Bieganowski received his M.Sc. degree in 2003. He has been working at the Faculty of Electrical Engineering, Computer Science and Telecommunications of the University of Zielona Góra since 2004. His scientific interests focus on logic synthesis and evolutionary algorithms.

Received: 11 December 2009
Revised: 14 June 2010
Re-revised: 26 August 2010