Int. J. Appl. Math. Comput. Sci., 2011, Vol. 21, No. 2, 401–412
DOI: 10.2478/v10006-011-0030-1

amcs

DESIGN OF MICROPROGRAMMED CONTROLLERS TO BE IMPLEMENTED IN FPGAs

REMIGIUSZ WIŚNIEWSKI *, ALEXANDER BARKALOV *, LARISA TITARENKO *, WOLFGANG A. HALANG **

* Faculty of Electrical Engineering, Computer Science and Telecommunications
University of Zielona Góra, ul. Podgórna 50, 65–246 Zielona Góra, Poland
e-mail: {r.wisniewski,a.barkalov,l.titarenko}@iie.uz.zgora.pl

** Faculty of Mathematics and Computer Science
University of Hagen, 58084 Hagen, Germany
e-mail: wolfgang.halang@fernuni-hagen.de

In the article we propose a new design method for microprogrammed controllers. The traditional structure is improved by modifying internal modules and connections. Such a solution allows reducing the total number of logic elements needed for implementation in programmable structures, especially Field Programmable Gate Arrays (FPGAs). Detailed results of experiments show that on the average the application of the proposed methods yields up to 30% savings as far as the destination device is considered.

Keywords: control unit, microprogrammed controller, field programmable gate array.

1. Introduction

A Control Unit (CU) is one of the main parts of any digital system (De Micheli, 1994; Maxfield, 2004; Gajski, 1996; Łuba, 2005). The traditional method of designing a digital system implements a CU as a Finite State Machine (FSM) (Łuba *et al*., 2009; Baranov, 1994; Adamski and Barkalov, 2006). Very often such a solution consumes many logic blocks of the implementation device, which could more effectively be used for other functions (Wiśniewski, 2009). It is known that in the case of the linear flow-chart, the microprogrammed controller (also known as the compositional microprogram control unit) consumes less hardware than controllers based on the traditional FSM model (Barkalov and Titarenko, 2009; Garcia-Vargas *et al*., 2007).

In a microprogrammed controller, the control unit is decomposed into two main parts. The first one is responsible for addressing microinstructions kept in the control memory. It is a simple finite state machine (Barkalov and Titarenko, 2009). The role of the second part is to hold and generate adequate microinstructions. Such a solution permits to minimise the number of logic elements used to implement the CU. Thus, wider areas of the target device are available for other modules of the designed system. The control unit's memory can be implemented using either logic elements or dedicated memory blocks on a chip (Wiśniewski, 2009). The rest of the system is realised by the logic blocks of the programmable device (for example, a field programmable gate array,) employed for implementation (Łuba, 2005). All logic functions are performed by Look-Up Tables (LUTs). As these LUTs have a limited number of inputs (Łuba, 2005; Maxfield, 2004), all Boolean functions (or the whole design) ought to be decomposed (Sentovich, 1993; Kania, 2004; Wiśniewska *et al*., 2007), which very often consumes an additional area on the destination FPGA.

This problem is addressed in the sequel by presenting a way to reduce the number of logic blocks required to implement microprogrammed controllers. Its main idea is to reduce the hardware amount by changing the structure of the controller. The internal blocks and connections of the control unit are modified to achieve better performance. The proposed method is compared with the traditional realisation technique of microprogrammed controllers. Detailed results of implementation and experimentation are given which indicate that the presen-

ted solution allows reducing the number of logic blocks by over 30%.

2. Microprogrammed controllers

Let a control algorithm be represented by a flow-chart Γ (Baranov, 1994; Barkalov and Titarenko, 2009) with a set of operational vertices $B = \{b_1, \ldots, b_K\}$ and a set of edges E. Each vertex $b_k \in B$ contains the microoperations $Y(b_k) \subseteq Y$, where $Y = \{y_1, \ldots, y_N\}$ is the set of microoperations. Each conditional vertex of the flow-chart contains one element from the set of logic conditions $X = \{x_1, \ldots, x_L\}$.

2.1. Main definitions. Let us introduce some definitions needed to explain the proposed methods.

Definition 1. *The Operational Linear Chain (OLC)* of the flow-chart Γ is a finite sequence of the operational vertices $\alpha_g = \langle b_{g1}, \ldots, b_{gFg} \rangle$ such that for any pair of adjacent components of the vector α_g there is an edge $\langle b_{gi}, b_{gi+1} \rangle \in E$, where i is the number of the component in the vector α_g $(i = 1, \ldots, F_g - 1)$.

Definition 2. The vertex $b_q \in B$ is called an *input of the OLC* α_g if there is an edge $\langle b_t, b_q \rangle \in B$, where b_t is either an initial or a conditional vertex of the flow-chart Γ, or an operational vertex not belonging to the OLC α_g.

Definition 3. The vertex $b_q \in B$ is named an *output of the OLC* α_g if there is an edge $\langle b_q, b_t \rangle$, where b_t is either a conditional or a final vertex of the flow-chart Γ, or an operational vertex not belonging to the OLC α_g.

Definition 4. The flow-chart Γ is called a *linear flow-chart* if the number of chains is at least twice less than the number of operator vertices (Barkalov and Titarenko, 2009).

2.2. Microprogrammed controller with mutual memory. Let D^g be a set of operational vertices included in the chain α_g, $C = \{\alpha_1, \ldots, \alpha_G\}$ a set of OLCs of the flow-chart Γ satisfying the condition

$$\begin{aligned} &D^g \cap D^q = \emptyset \quad (g \neq q, \quad g, q \in \{1, \ldots, G\}), \\ &B = D^1 \cup D^2 \cup \cdots \cup D^G, \\ &D^g \neq \emptyset \quad (g \in \{1, \ldots, G\}), \end{aligned} \tag{1}$$

and let natural addressing of microinstructions be executed for each α_g:

$$A(b_{gi+1}) = A(b_{gi}) + 1 \quad (i \in \{1, \ldots, F_{g-1}\}), \tag{2}$$

where $A(b_g)$ is the address of the microinstruction corresponding to the vertex $b_g \in B$. Then the flow-chart Γ can be interpreted as a Compositional Microprogrammed Control Unit (CMUC) with mutual memory denoted by U_{MM} (Fig. 1).

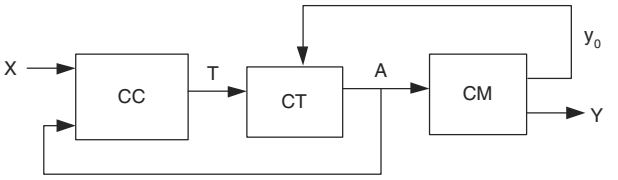

Fig. 1. Structure of a microprogrammed controller with mutual memory.

In this control unit, the combinational circuit CC is responsible for generating excitation functions for the counter CT, which keeps an address of microinstructions. The variables $A_r \in A$ are used to represent addresses $A(b_k)$, $b_k \in B$. Microinstructions are kept in the control memory CM, and each word (microinstruction) has $N+2$ bits in the case of unitary encoding of microoperations (Barkalov and Titarenko, 2009). One of the additional bits is used to keep a variable y_0 to organise the addressing mode (2). The second additional bit keeps a variable y_K to organise the microinstruction fetching from CM. For simplicity, this variable is not shown in all figures in this paper.

The controller operates as follows. At the beginning, the counter is set to the value that corresponds to the initial state of the FSM, which is equal to the address of the first microinstruction of the control algorithm to be implemented. If transitions are executed inside some chain $\alpha_g \in C$, then $y_0 = 0$. This causes the CT to be incremented and prohibits changing the current state of the control unit. When the output of $\alpha_g \in C$ is reached, $y_0 = 1$ and the circuit CC forms the excitation function for the counter

$$T = f(X, A). \tag{3}$$

This function forms the code of the state of transition and the address of the input of the next OLC $\alpha_g \in C$ as well. If the controller reaches an address of the microinstruction $Y(b_k)$ such that $\langle b_k, b_E \rangle \in E$, then $y_K = 1$. Thus, the operation of the CMCU U_{MM} is finished.

2.3. Example of synthesising a microprogrammed controller with mutual memory. To elucidate the idea of the CMCU with mutual memory, a method for synthesising such a controller is now illustrated with a simple example. Figure 2 shows a hypothetical algorithm of the control unit U_1. Here the symbol U_1 stands for the CMCU U_{MM} in our example. There are 11 operational vertices $B = \{b_1, \ldots, b_{11}\}$ and three conditional vertices with conditions from the set $X = \{x_1, x_2, x_3\}$ in the flow-chart Γ_1. Thus, the circuit should generate 11 microinstructions that consist of five microoperations $Y = \{y_1, \ldots, y_5\}$.

In order to design a microprogrammed controller with mutual memory, first the set C of operational linear

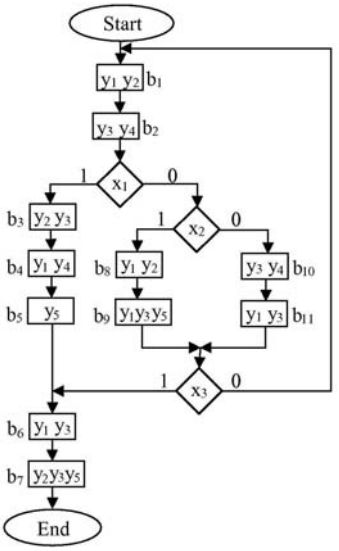

Fig. 2. Flow-chart Γ_1.

chains ought to be formed (Fig. 3). In the presented example, there are four OLCs $C = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, where $\alpha_1 = \langle b_1, b_2\rangle$, $\alpha_2 = \langle b_3, \ldots, b_7\rangle$, $\alpha_3 = \langle b_8, b_9\rangle$, and $\alpha_4 = \langle b_{10}, b_{11}\rangle$. All OLCs, except for α_2, have one input: for α_1 it is the vertex b_1 and for α_3 and α_4 they are b_3 and b_4, respectively. The OLC α_2 has two inputs: the vertices b_3 and b_6. Therefore, the set of inputs contains five elements: $I = \{I_1^1, I_2^1, I_2^2, I_3^1, I_4^1\}$, where $I_1^1 = b_1$, $I_2^1 = b_3$, $I_2^2 = b_6$, $I_3^1 = b_8$ and$I_4^1 = b_{10}$. Each OLC may have only one output. Thus, there are four outputs in the set of OLCs: $O = \{O_1, \ldots, O_4\}$, where $O_1 = b_2$, $O_2 = b_7$, $O_3 = b_9$ and $O_4 = b_{11}$.

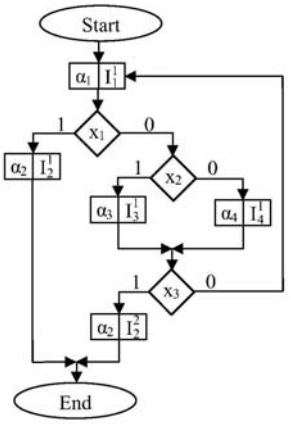

Fig. 3. OLC flow-chart of the CMCU U_1.

In the next step of the design process, the content of the control memory is formed. To this end, the addresses of all microinstructions have to be encoded. For a control unit with mutual memory, the encoding method is not important. Therefore, according to (2), natural binary codes are used. As there are 11 operational vertices in the flow-chart Γ_1, the microinstructions are encoded using four bits. In the presented example, the microinstructions are addressed as $A(b_0) = 0000$, $A(b_1) = 0001$, ..., $A(b_{11}) = 1010$.

Each microinstruction written at the vertex b_k consists of microoperations that are written in this vertex. Two additional microoperations are necessary for the CMCU to function properly: y_0 and y_K. The first one is set up ($y_0 = 1$) if the vertex b_k belongs to the set of outputs O. Otherwise, $y_0 = 0$. In the example, y_0 is produced by the vertices b_2, b_7, b_9 and b_{11}. The microoperation y_K is equal to 1 only if the vertex b_k is connected with the final vertex of the flow-chart. For the flow-chart Γ_1, y_k is set at the vertex b_7 only.

Next, microinstructions are encoded and the table of the control memory content is formed. Table 1 represents the content of CM for the control unit U_1.

Table 1. Content of the control memory of the CMCU U_1.

Vertex	Address	Microinstruction						
		y_0	y_1	y_2	y_3	y_4	y_5	y_K
b_1	0000	0	1	1	0	0	0	0
b_2	0001	1	0	0	1	1	0	0
b_3	0010	0	0	1	1	0	0	0
b_4	0011	0	1	0	0	1	0	0
b_5	0100	0	0	0	0	0	1	0
b_6	0101	0	1	0	1	0	0	0
b_7	0110	1	0	1	1	0	1	1
b_8	0111	0	1	1	0	0	0	0
b_9	1000	1	1	0	1	0	1	0
b_{10}	1001	0	0	0	1	1	0	0
b_{11}	1010	1	1	0	1	0	0	0

To determine the excitation function T for the counter, the table of transitions of the CMCU U_1 has to be formed. This table describes transitions between all operational linear chains depending on input values (set of operational vertices X). In the presented example, the table of transitions (Table 2) has $H = 8$ lines.

Based on the address $SA(O_g)$ (which is represented by the set of variables $A = \{a_1, \ldots, a_4\}$) and on the set of conditional vertices X, the counter's excitation function T is formed:

$$\begin{aligned} t_4 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot\overline{x_2},\\ t_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1}\cdot x_2 + a_4\cdot\overline{a_3}\cdot\overline{a_1}\cdot x_3,\\ t_2 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot(x_1 + \overline{x_1}\cdot x_2),\\ t_1 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{a_2}\cdot a_1\cdot\overline{x_1} + a_4\cdot\overline{a_3}\cdot\overline{a_1}\cdot x_3. \end{aligned} \tag{4}$$

Table 2. Table of transitions of the CMCU U_1.

O_g	$SA(O_g)$				X_h	I_j^t	$K(I_j^t)$				T	h
	a_4	a_3	a_2	a_1			t_4	t_3	t_2	t_1		
O_1	0	0	0	1	x_1	I_2^1	0	0	1	0	t_2	1
O_1	0	0	0	1	$\overline{x_1}\,x_2$	I_3^1	0	1	1	1	$t_3\,t_2\,t_1$	2
O_1	0	0	0	1	$\overline{x_1}\,\overline{x_2}$	I_4^1	1	0	0	1	$t_4\,t_1$	3
O_2	0	1	1	0	–	–	–	–	–	–	–	4
O_3	1	0	0	0	x_3	I_2^2	0	1	0	1	$t_3\,t_1$	5
O_3	1	0	0	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	6
O_4	1	0	1	0	x_3	I_2^2	0	1	0	1	$t_3\,t_1$	7
O_4	1	0	1	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	8

Now, the CMCU U_1 can easily be prototyped using hardware description languages such as Verilog (Lee, 1999; Thomas and Moorby, 2002) or VHDL (Brown and Vranesic, 2000; Zwolinski, 2000). Based on such a description, the CMCU can be logically synthesised and finally implemented in an FPGA.

A relevant example was prepared and implemented using a type XC2VP30 FPGA of the Virtex-II Pro family of Xilinx. Figure 4 shows a simplified technological diagram of the controller. Initially, the diagram was generated after logic synthesis by the Xilinx XST tool. It was modified to clarify the logic structure of circuit U_1. Here 10 LUTs corresponding to the combinational circuit were replaced by one block. Similarly, four LUTs and four flip-flops forming the counter are represented by two further blocks. Being a Xilinx primitive, *FDC* represents a D-type flip-flop with asynchronous reset. Additionally, the main nets were named (in the example T, A) to show the similarity to the logic diagram.

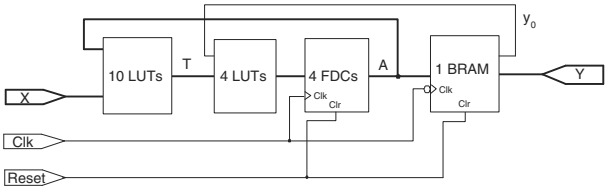

Fig. 4. Technological structure of the CMCU U_1.

Two blocks of the microprogrammed controller U_1 are synchronous: counter and control memory. Therefore, the clock signal *Clk* ought to be delivered to them. The counter is triggered by the rising edge of the clock signal. Because of feedback signals, however, the control memory is active on the falling edge of *Clk*. Now, an address of a microinstruction is formed on a positive edge, while outputs are generated when the clock signal goes low. Needless to say that critical timing paths should be checked to avoid timing skews in the circuit (placement and timing paths are automatically verified by Xilinx tools during logical implementation of the design).

The circuit of the CMCU U_1 took 14 LUT elements and one dedicated memory block of FPGA resources. In contrast, when prepared as a traditional finite state machine, the controller required 14 LUT elements and one dedicated memory block as well (here microinstructions were also implemented using dedicated memory). This simple example reveals that a controller designed as a CMCU with mutual memory may not give better results than the equivalent FSM-based circuit.

The results achieved with more tests (as presented in Section 5 in detail) showed for controllers interpreting a linear flow-chart that CMCUs with mutual memory require fewer logic blocks than traditional FSMs. Although the benefit is rather low (about 9%), these results were an inspiration to search for new design ideas of control units. The aim of the research was to reduce the number of logic elements required to implement controllers using programmable devices. The next sections show how to improve microprogrammed controllers with mutual memory.

3. Microprogrammed controller with identification of outputs

In the microprogrammed controller with mutual memory, as shown in the previous section, the microinstruction address generated by the counter is used to recognise the controller's current operational linear chain. Based on this code, the module CC executes the system function according to (3). Here, the full address is used as a feedback function—in our example we used $|A| = 4$ bits. However, since there are only $|C| = 3$ OLCs in the controller, they may be encoded with only R_{OI}=$\lceil \log_2 3 \rceil = 2$ bits. Such a solution may be possible, depending on the encoding of each OLC's output.

If we encode microinstructions as shown in Table 3, we see that the OLC outputs correspond to the addresses $A(O_1) = 0001$, $A(O_2) = 0110$, $A(O_3) = 1000$ and $A(O_1) = 1100$. Now each output can be recognised with two major bits of its address. Finally, the OLC outputs are encoded as follows: $K(O_1) = 00$, $K(O_2) = 01$, $K(O_3) = 10$ and $K(O_4) = 11$. There are only two variables used to represent each code: $Q = \{a_3, a_4\}$, where

$Q \in A$. This means that the set of feedback variables used to identify the current state of the controller is reduced to the minimum. A combinational circuit generates the function T for the counter (Wiśniewski *et al*., 2006; Barkalov *et al*., 2006):

$$T = f(X, Q), \tag{5}$$

where $Q \subseteq A$, $|Q| = R_{OI}$, $Q = \{Q_1, \ldots, Q_{R_{OI}}\}$.

Table 3. Microinstruction addressing in the microprogrammed controller U_2.

Vertex	Address	Comment
b_1	0000	I_1^1
b_2	0001	O_1
b_3	0010	I_2^1
b_4	0011	–
b_6	0101	I_2^2
b_7	0110	O_2
b_8	0111	I_3^1
b_9	1000	O_3
b_{10}	1011	I_4^1
b_{11}	1100	O_4

Figure 5 illustrates the structure of the microprogrammed controller with output identification. As already mentioned, the main idea of the device presented is to use the part Q of the address A to identify the control unit's internal states.

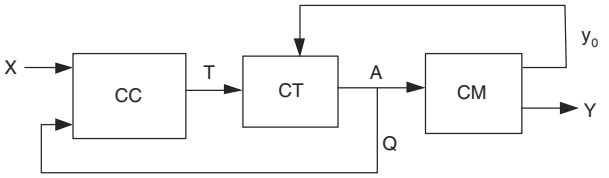

Fig. 5. Structure of the CMCU with output identification.

3.1. Synthesis of the CMCU with output identification. The method proposed to synthesise the CMCU with output identification includes the following steps:

1. *Formation of the OLC set.* The set of operational linear chains is created. For each OLC, its output and all inputs are determined. There are M_2 operational linear chains, and the length of the longest one is specified by the value M_1. The total number of microinstructions is represented by the parameter M_3.

2. *Addressing microinstructions and encoding OLC outputs.* Let $Q \subseteq A$ be a set of variables sufficient for one-to-one identification of the OLC, $\alpha_g \in C$ and $R_{OI} = |Q|$. Addressing the CMCU's microinstructions is carried out as follows:

   (a) At the beginning, all microinstructions are encoded using natural binary codes.

   (b) The value of R_{OI} is set to $R_{OI} = R_2$, where $R_2 = \lceil \log_2 M_2 \rceil$.

   (c) The addressing table is created. It has $2^{R_{OI}}$ columns marked by R_{OI} major address bits and $2^{R_3 - R_{OI}}$ lines marked by $R_3 - R_{OI}$ minor address bits. Here, $R_3 = \lceil \log_2 M_3 \rceil$.

   (d) If outputs of two different OLCs $\alpha_i, \alpha_j \in C$ are located in the same column and none of the outputs is connected with the final vertex of the flow-chart, then the information is shifted to the right starting from the first vertex of the OLC α_j ($j > i$). The table's cells releasing entries are filled with the symbol '$*$'. This operation is performed until the outputs O_i and O_j are in different columns of the table.

   (e) If the outputs of all OLCs are identified one-to-one by R_{OI} bits, then the algorithm moves on to point (g).

   (f) If the address of any vertex is beyond the actual addressing space, then $R_{OI} := R_{OI} + 1$. Next, the algorithm is repeated from point (c) on.

   (g) End.

   Finally, all microinstructions are encoded. Now each microinstruction's code is formed as concatenation of major (columns) and minor (lines) addresses of the created table. Outputs of OLCs are encoded using only major address bits. The outcome of this encoding is further used to form the transition table of the microprogrammed controller.

3. *Formation of the control memory content.* The content of control memory is formed. Addresses of microinstructions are created according to the algorithm presented in the previous step.

4. *Formation of the transition table of the CMCU U_{OI} and the counter's excitation function.* At this stage, the table of transitions between the OLCs is created. It contains the columns O_g, $MA(O_g)$, X_h, I_j^t, $K(I_j^t)$, T, h, where

   - O_g is the output of the chain from which the transition is executed;
   - $MA(O_g)$ is the major part of the output's O_g address; this address was calculated at Stage 2;
   - X_h is the input signal causing transition $\langle O_g, I_t^j \rangle$; it is equal to the conjunction of the elements from the set X;
   - I_j^t is the input of the chain $\alpha_j \in C$ in which the transition is executed;

- $K(I_j^t)$ is the address of the input I_j^t;
- T is the set of variables forming the excitation function for the counter;
- h is the number of the transition ($h = 1, \ldots, H$).

Based on this table, the counter's excitation function T is formed as

$$T_r = \bigvee_{h=1}^{H} C_{rh} E_g^h X_h \quad (r \in \{1, \ldots, R_{OI}\}). \quad (6)$$

Here C_{rh} is a Boolean variable that is equal to 1 if and only if the function T_r is written in the h-th line of the table of transitions; E_g^h is a conjunction of the internal variables $Q_r \in Q$ corresponding to the address $MA(O_g)$ of the output O_g from the h-th line of the table of transitions.

5. *Implementation of CMCU* U_{OI}. This step is executed in the same manner as was shown during the design process of the CMCU U_{MM}. The combinational circuit and the counter are implemented using LUT elements while the control memory is realised with dedicated memory blocks of FPGAs.

3.2. Example of synthesising the microprogrammed controller with identification of outputs. To elucidate the idea of OLC encoding, the design process of the CMCU U_{OI} with output identification is illustrated by an example. Once again, the flow-chart Γ_1 is used as initial description of controller U_2. There are $M_3 = 11$ operational vertices and $M_2 = 4$ operational linear chains. The longest OLC is α_2 containing $M_1 = 5$ elements. According to the algorithm of microinstruction addressing, the initial value of variable R_{OI} is equal to $R_2 = \lceil \log_2 M_2 \rceil = 2$. Thus, at the beginning, the table of addressing has $2^{R_{\mathrm{OI}}} = 2$ columns and $2^{R_3 - R_{OI}} = 2$ lines (Fig. 6).

$a_1\,a_2$ \ $a_3\,a_4$	00	01	10	11
00	$b_1 = I_1^1$	b_5	$b_9 = O_3$	*
01	$b_2 = O_2$	$b_6 = I_2^2$	$b_{10} = I_4^1$	*
10	$b_3 = I_2^1$	$b_7 = O_2$	$b_{11} = O_4$	*
11	b_4	$b_8 = I_3^1$	*	*

Fig. 6. Initial table of addressing.

Initially, all addresses of microinstructions are encoded in natural binary code. In the presented example, the outputs O_3 of α_3 and O_4 of α_4 are located in the same column. Since neither O_3 nor O_4 are connected to the final vertex of the flow-chart Γ_1, all components that have higher addresses than output O_3 are shifted. This movement is performed while output O_4 is in the same column as O_3.

Figure 7 presents the table after the shift operation. Now each OLC output is located in a different column, and there are no vertices beyond the addressing space. This means that all addresses are encoded and the algorithm is finished.

$a_1\,a_2$ \ $a_3\,a_4$	00	01	10	11
00	$b_1 = I_1^1$	b_5	$b_9 = O_3$	$b_{11} = O_4$
01	$b_2 = O_2$	$b_6 = I_2^2$	*	*
10	$b_3 = I_2^1$	$b_7 = O_2$	*	*
11	b_4	$b_8 = I_3^1$	$b_{10} = I_4^1$	*

Fig. 7. Table of addressing after shift operations.

In the next step, the table of transitions of the CMCU U_2 is created. Here the symbol U_2 stands for the CMCU U_{OI} as implemented in our example. The table is similar to the one created for the CMCU with mutual memory, although now there are only two major bits of the whole address used as OLC output identification (Tab. 4).

Based on the address $MA(O_g)$ (represented by the set of variables $Q = \{a_3, a_4\}$) and the set of logical conditions X, the counter's excitation function T is formed:

$$\begin{aligned} t_4 &= \overline{a_4} \cdot \overline{a_3} \cdot \overline{x_1} \cdot \overline{x_2}, \\ t_3 &= \overline{a_4} \cdot \overline{a_3} \cdot \overline{x_1} \cdot x_2 + a_4 \cdot x_3, \\ t_2 &= \overline{a_4} \cdot \overline{a_3}, \\ t_1 &= \overline{a_4} \cdot \overline{a_3} \cdot \overline{x_1} + a_4 \cdot x_3. \end{aligned} \quad (7)$$

The microprogrammed controller U_2 can be prototyped using Hardware Description Languages (HDLs). In comparison to the excitation function formed for the controller with mutual memory, T contains fewer variables and shorter equations. Therefore, it is expected that the CMCU U_2 should consume fewer logic elements than the CMCU U_1. In fact, implementing the controller on an FPGA showed that the CMCU U_2 with output identification requires 11 LUT elements (Fig. 8), which means a reduction by 21% in comparison with the microprogrammed controller U_1 with mutual memory.

Table 4. Transition table of the CMCU U_2.

O_g	$MA(O_g)$ a_4	a_3	X_h	I_j^t	$K(I_j^t)$ t_4	t_3	t_2	t_1	T	h
O_1	0	0	x_1	I_2^1	0	0	1	0	t_2	1
O_1	0	0	$\overline{x_1}\,x_2$	I_3^1	0	1	1	1	$t_3\,t_2\,t_1$	2
O_1	0	0	$\overline{x_1}\,\overline{x_2}$	I_4^1	1	0	1	1	$t_4\,t_2\,t_1$	3
O_2	0	1	–	–	–	–	–	–	–	4
O_3	1	0	x_3	I_2^2	0	1	0	1	$t_3\,t_1$	5
O_3	1	0	$\overline{x_3}$	I_1^1	0	0	0	0	–	6
O_4	1	1	x_3	I_2^2	0	1	0	1	$t_3\,t_1$	7
O_4	1	1	$\overline{x_3}$	I_1^1	0	0	0	0	–	8

4. Microprogrammed controller with identification of outputs and a function decoder

Figure 9 shows the microprogrammed controller with a function decoder. The main idea for improving the structure of the CMCU with output identification is to reduce the number of logic blocks of the target FPGA by employing an additional block (function decoder), which may be implemented using dedicated memories. As a result, fewer LUT elements are needed to realise the control unit as compared to the CMCUs shown in previous sections.

4.1. Main idea of the method. In the CMCU U_{OD}, the variables forming the counter's excitation function are encoded with the minimum number of bits. To this end, all inputs of operational linear chains ought to be encoded. Moreover, an address of each microinstruction is encoded and recognised with Q bits, according to (5). Now the module CC generates a function Z:

$$Z = f(X, Q), \tag{8}$$

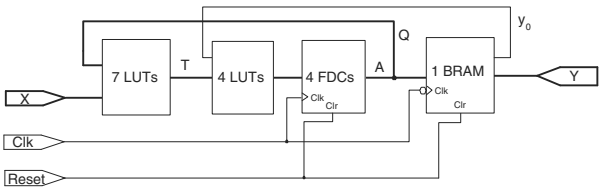

Fig. 8. Technological structure of the microprogrammed controller U_2.

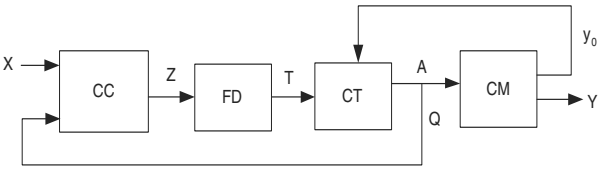

Fig. 9. Structure of CMCU with output identification and a function decoder.

which contains the encoded addresses $E(I)$ of all inputs in the set of OLCs. They are further decoded by the block FD, which indicates the proper code for the counter:

$$T = f(Z), \tag{9}$$

where T means the set of variables forming the counter's excitation function.

The number of bits required to encode all inputs can be calculated as $R_Z = \lceil \log_2 M_Z \rceil$, where $M_Z = |I|$ is equal to the number of all inputs in the set of OLCs.

The presented solution permits to reduce the number of outputs generated by the circuit CC. The additional block of the function decoder is implemented with dedicated FPGA memories. Therefore, the number of logic elements needed to implement the entire controller is reduced.

4.2. Synthesising the microprogrammed controller with identification of outputs and a function decoder. The proposed design method for the CMCU U_{FD} includes the following steps:

1. *Forming the set of OLCs and encoding their inputs.* The set of OLCs is formed in the same manner as shown during the synthesis of the microprogrammed controller with mutual memory. Next, the addresses A of all microinstructions are calculated. The encoding style is not important, so natural binary codes may be used. Finally, the addresses $K(I_j^t)$ of all inputs of the set of OLCs are encoded with the minimum number of bits R_Z. Now each input has a unique code $E(I_j^t)$.

2. *Addressing microinstructions and encoding OLC out puts.* Addresses of microinstructions are represented using the algorithm shown in the previous section. The outputs of OLCs are encoded employing major address bits only. The outcome of the encoding is further used in forming the microprogrammed controller's table of transitions.

3. *Formation of the control memory content.* In accordance with the addresses calculated in the previous stage, the content of the control memory is prepared.

4. *Formation of the CMCU's transition table.* The table of transitions is the basis to form the system (8) and to synthesise the circuit CC. This table contains only transitions for such OLCs whose outputs are not connected to the final vertex of the flow-chart. The table of transitions contains the columns O_g, $MA(O_g)$, X_h, I_j^t, $E(I_j^t)$, Z, h, where

   - O_g is the output from which the transition is executed;
   - $MA(O_g)$ is the major part of the output's O_g address; this address was calculated at the stage of microinstruction addressing;
   - X_h is the input signal causing transition $\langle O_g, I_t^j \rangle$; it is equal to the conjunction of the elements from set X;
   - I_j^t is the input of the chain $\alpha_j \in C$ to which the transition is executed;
   - $E(I_j^t)$ is the address of input I_j^t;
   - Z is the set of variables forming the decoder's excitation function;
   - $h \in \{1, \ldots, H\}$ is the number of the transition.

   Based on the transition table, the excitation function Z can be determined. The system (8) is represented as

$$z_r = \bigvee_{h=1}^{H} C_{rh} F_g^h X_h \quad (r \in \{1, \ldots, R_1\}), \qquad (10)$$

   where C_{rh} is a Boolean variable equal to 1 if and only if the function z_r is written in the h-th line of the table of transitions; F_g^h is a conjunction of the internal variables $a_r \in A$ corresponding to the address $SA(O_g)$ of the output O_g from the h-th line of the table of transitions.

5. *Formation of the function decoder's table.* Based on the code $E(I_j^t)$, the function decoder generates the proper address $K(I_j^t)$ of the OLC input. The set of addresses $K(I_j^t)$ form the counter's excitation function T. The function decoder's table contains the columns I_j^t, $K(I_j^t)$, $E(I_j^t)$, T, m, where

   - I_j^t is the input of the chain $\alpha_j \in C$;
   - $E(I_j^t)$ is the encoded address of input I_j^t;
   - $K(I_j^t)$ is the code of input I_j^t;
   - T is the set of variables forming the counter's excitation function;
   - m is the consecutive line in the truth-table of the function decoder ($m = 1, \ldots, M$).

   Based on this table, the circuit of the function decoder can be implemented with dedicated memory blocks. The code $E(I_j^t)$ represents inputs and $K(I_j^t)$ represents outputs of the function decoder. The volume of memory required to implement the function decoder can be calculated as S_{FD}=$R_1 \cdot 2^{R_Z}$, where R_1 counts the number of variables forming the counter's excitation function and R_Z stands for the number of bits required for OLC input encoding.

6. *Implementation of the CMCU U_{OD}.* The main advantage of the microprogrammed controller with output identification and a function decoder is the possibility to implement both blocks (FD and CM) with dedicated memory blocks. Moreover, due to output identification, the number of feedback functions for the combinational circuit decreases in comparison with the CMCU U_{MM}. Therefore, implementation of the CMCU U_{OD} consumes the least logic elements of programmable devices in comparison with the CMCUs U_{MM}, U_{FD} and U_{OI}. It should be pointed out, however, that the presented controller uses at least two dedicated memory blocks of an FPGA.

4.3. Example of synthesising a CMCU with identification of outputs and a function decoder. To illustrate the synthesis of the microprogrammed controller U_{OD}, the flow-chart Γ_1 is used as its initial description. Let us denote by U_3 the CMCU U_{OD} as in this example. The prototyping process of the CMCU U_3 with output identification and a function decoder is a conjunction of the designs of the CMCUs U_1 and U_2. At the beginning, the set of OLCs is formed and all OLCs inputs are encoded. As was presented in the previous sections, there are four OLCs which have five inputs. Thus, OLC inputs may be encoded using $|Z| = 3$ bits. In this example, a natural binary code is used: $E(I_1^1) = 000$, $E(I_1^2) = 001$, $E(I_2^2) = 010$, $E(I_1^1) = 011$ and $E(I_1^1) = 100$.

At the next stage, the addressing of microinstructions and the encoding of OLC outputs are to be performed. According to the algorithm presented in Section 3.1, microinstructions corresponding to the vertices $b_1, \ldots, b_9$ are addressed consecutively in a natural binary code: $A(b_1) = 0000$, $A(b_2) = 0001$, $A(b_3) = 0010$, $\ldots$, $A(b_9) = 1000$. The addresses of the last two microinstructions are shifted. Thus their codes are $A(b_{10}) = 1011$ and $A(b_{11}) = 1100$. The outputs of OLCs are encoded with $|Q| = 2$ major address bits, and hence $\mathrm{MA}(O_1) = 00$, $\mathrm{MA}(O_2) = 01$, $\mathrm{MA}(O_3) = 10$ and $\mathrm{MA}(O_4) = 11$. The content of the control memory is shown in Table 5.

Next, the transition table of the CMCU is prepared. It contains transitions from output O_i (encoded using

$Q \subset A$ bits) to input I_j^t (encoded using Z bits). Table 6 represents the transition table for the CMCU U_3. From the table of transitions, the following excitation function Z for the function decoder is formed:

$$\begin{aligned} z_3 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot\overline{x_2}, \\ z_2 &= \overline{a_4}\cdot\overline{a_3}\cdot\overline{x_1}\cdot x_2 + a_4\cdot x_3, \\ z_1 &= \overline{a_4}\cdot\overline{a_3}\cdot(x_1 + \overline{x_1}\cdot x_2). \end{aligned} \tag{11}$$

In order to generate a proper excitation function for the counter, a table of the function decoder has to be prepared. Table 7 shows the content of the function decoder for the CMCU U_3.

The block FD may be implemented either using dedicated memories or with logic blocks of an FPGA. In the case of LUT elements realisation, the minimised excitation function T is additionally formed:

$$\begin{aligned} t_4 &= z_3\cdot\overline{z_2}\cdot\overline{z_1}, \\ t_3 &= \overline{z_3}\cdot z_2, \\ t_2 &= \overline{z_3}\cdot z_1, \\ t_1 &= \overline{z_3}\cdot z_2 + z_3\cdot\overline{z_2}\cdot\overline{z_1}. \end{aligned} \tag{12}$$

Finally, the controller may be designed with HDL languages and implemented in a programmable device. An implementation of the microprogrammed controller U_3 in an FPGA is schematically shown in Fig. 10. As expected, the CMCU U_3 requires the fewest logic blocks of the device among all controllers previously presented. The conjunction of OLC output identification and applying the function decoder resulted in a reduction in LUT elements used to 10. This means that the amount of hardware required to implement the initial microprogrammed controller with mutual memory was decreased by 26%.

Table 5. Control memory content of the CMCU U_3.

Vertex	Address	Microinstruction						
		y_0	y_1	y_2	y_3	y_4	y_5	y_K
b_1	0000	0	1	1	0	0	0	0
b_2	0001	1	0	0	1	1	0	0
b_3	0010	0	0	1	1	0	0	0
b_4	0011	0	1	0	0	1	0	0
b_5	0100	0	0	0	0	0	1	0
b_6	0101	0	1	0	1	0	0	0
b_7	0110	1	0	1	1	0	1	1
b_8	0111	0	1	1	0	0	0	0
b_9	1000	1	1	0	1	0	1	0
b_{10}	1011	0	0	0	1	1	0	0
b_{11}	1100	1	1	0	1	0	0	0

Table 6. Table of transitions of the CMCU U_3.

O_g	$MA(O_g)$		X_h	I_j^t	$E(I_j^t)$			Z	h
	a_4	a_3			z_3	z_2	z_1		
O_1	0	0	x_1	I_2^1	0	0	1	z_1	1
O_1	0	0	$\overline{x_1}\cdot x_2$	I_3^1	0	1	1	$z_2\, z_1$	2
O_1	0	0	$\overline{x_1}\cdot\overline{x_2}$	I_4^1	1	0	0	z_3	3
O_2	0	1	–	–	–	–	–	–	4
O_3	1	0	x_3	I_2^2	0	1	0	z_2	5
O_3	1	0	$\overline{x_3}$	I_1^1	0	0	0	–	6
O_4	1	1	x_3	I_2^2	0	1	0	z_2	7
O_4	1	1	$\overline{x_3}$	I_1^1	0	0	0	–	8

Table 7. Table of the function decoder for the CMCU U_3.

I_j^t	$E(I_j^t)$			$K(I_j^t)$				T	m
	z_3	z_2	z_1	t_4	t_3	t_2	t_1		
I_1^1	0	0	0	0	0	0	0	–	1
I_2^1	0	0	1	0	0	1	0	t_2	2
I_2^2	0	1	0	0	1	0	1	$t_3\, t_1$	3
I_3^1	0	1	1	0	1	1	1	$t_3\, t_2\, t_1$	4
I_4^1	1	0	0	1	0	0	1	$t_4\, t_1$	5

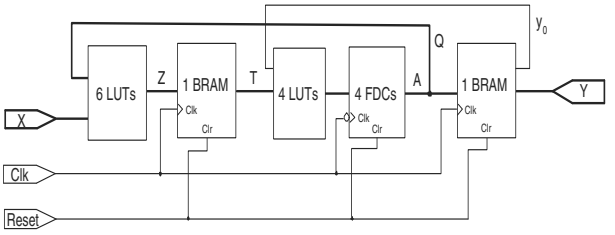

Fig. 10. Technological structure of the CMCU U_3.

5. Results of evaluations and experiments

Experiments have been performed to prove the effectiveness of the proposed methods. First, the tool for automatic synthesis of CMCUs will be described. Such a system is absolutely indispensable in designing bigger microprogrammed controllers. Next, formal verification of CMCUs prepared with different methods will be reported. Finally, experimental results and a detailed analysis of the obtained values will be given.

5.1. CAD tool for automatic synthesis of CMCUs (ATOMIC). To automate the design process of microprogrammed controllers, a dedicated CAD tool was developed to perform *A*u*TOM*atic synthes*I*s of *C*MCUs (ATOMIC). Based on the description of a controller as a flow-chart, ATOMIC produces code in a hardware description language (Verilog). Such code is ready for logic synthesis and subsequent implementation in an FPGA. The tool's main features are shown in this section. In the work of Wiśniewski (2009), input and output data formats, as well as switches and parameters are described in detail.

There are three major modules constituting ATOMIC (Fig. 11). The first module (*fc2olc*) analyses the structure of a given flow-chart and produces a set of operational

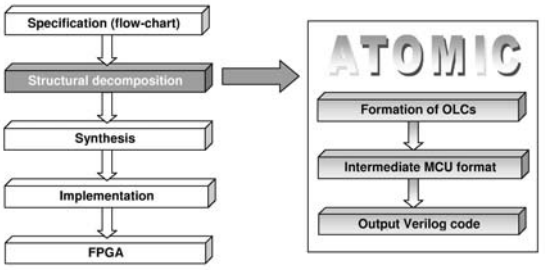

Fig. 11. Structure of ATOMIC.

linear chains. This step is common for all implemented methods. The second module (*olc2mcu*) is based on the description of OLCs, and the chosen method performs the structural decomposition process. All required data (excitation functions, description of control memory, etc.) are stored using an intermediate format. Such a format may be the basis for various ways to describe a CMCU. For example, Verilog or VHDL code may very easily be produced. The last module of ATOMIC (*mcu2verilog*) generates direct descriptions of CMCUs using Verilog HDL. These descriptions are ready for logic synthesis and implementation.

ATOMIC was designed as a module-based tool in order to provide a high performance. At each stage, the description of a controller prototyped may be changed. Furthermore, once prepared, an OLC description may commonly be used as input for all three synthesis methods implemented.

A very important feature is the possibility to use external tools for further analysis. Each excitation function produced by the *olc2mcu* module may be decomposed with appropriate other systems like SIS or DEMAIN (Łuba, 2005). Therefore, both structural and functional decompositions can be used in the prototyping flow of a microprogrammed controller. A control unit is initially decomposed with structural procedures, and then excitation functions produced for internal blocks of the microprogrammed controller are optimised by functional decomposition. Such a solution preserves the structure of the CMCU, which leads to the possibility of partially reconfiguring the controller (Wiśniewski, 2005).

5.2. Library of test modules. The presented design methods for microprogrammed controllers were verified with over 100 test modules (benchmarks). Each test module was prepared in a text format containing a description of the tested CMCU as a flow-chart. The library of test modules contains benchmarks taken from the works of Barkalov and Titarenko (2009), as well as Wiśniewski (2009). Most of them relate to real applications (e.g., traffic light controller or arithmetic operations), whereas only some comprise artificial test cases.

5.3. Verification of the prepared methods. The functionality of designed the CMCUs was verified with a software simulator (here, Active HDL of Aldec and ModelSim of Mentor Graphics). Simulations were performed for each synthesis method. Each module was verified similarly. First, Verilog code was generated for each synthesis method using ATOMIC. Next, controllers were simulated and their functionality was verified. Finally, the functionality of control units designed with four different methods (traditional Moore FSM, CMCU with mutual memory, CMCU with identification of outputs, CMCU with a function decoder and identification of outputs) were compared. Verification of all controllers resulted in correct values, i.e., all CMCUs prepared with the proposed methods provide proper functionality.

5.4. Results of experiments. As has already been mentioned, all synthesis methods were verified by over 100 benchmarks. Additionally, for each test, an FSM model was prepared. These automata were created according to the rules presented by Thomas and Moorby (2002), Bukowiec (2009) as well as Barkalov, Titarenko and Chmielewski (2007). All FSMs were prepared in such a way that during FPGA implementation all microoperations were realised with dedicated memory blocks.

The prototyping process for each benchmark was similar. Based on its flow-chart description (*.fc* file), a controller was structurally decomposed with all design methods presented here. Additionally, an equivalent FSM was produced. The generated Verilog codes were finally synthesised and implemented with the Xilinx XST tool.

Table 8 presents average results of CMCU implementations designed with the particular synthesis method in comparison with the Moore FSM and the CMCU with mutual memory. As the target, the FPGA XC2VP30 of the Xilinx Virtex-II Pro family was selected. The device contains 27392 flip-flops, 27392 LUTs (13696 slices) and 136 dedicated memory blocks (block RAMs). Detailed results of performed experiments can be found at `http://www.uz.zgora.pl/~rwisniew/badania/results_amcs_2010.html`.

5.5. Analysis of experimental results. Detailed analysis of the results obtained proved the effectiveness of the proposed methods. The designs $CMCU_{OI}$ with identification of outputs and $CMCU_{OD}$ with identification of outputs and a function decoder require less logic blocks than the controller with mutual memory. Moreover, both methods permit even to reduce the number of slices and LUTs in comparison with traditional FSMs by over 40% (in the case of the $CMCU_{\mathrm{OD}}$). The investigations also

Table 8. Averaged experimental results.

	FPGA resources	Design method FSM	MM	OI	OD
Comparison with the FSM	Slices	100%	91%	76%	60%
	FF	100%	100%	102%	108%
	LUTs	100%	91%	78%	60%
	BRAMs	100%	100%	102%	126%
Comparison with the CMCU with mutual memory	Slices	110%	100%	84%	68%
	FF	100%	100%	102%	108%
	LUTs	110%	100%	86%	68%
	BRAMs	100%	100%	102%	126%

FSM: controller realised as the FSM, MM: controller realised as the CMCU with mutual memory,
OI: controller realised as the CMCU with identification of outputs,
OD: controller realised as the CMCU with identification of outputs and a function decoder.

show that the $CMCU_{\mathrm{OD}}$ uses much less logic than the $CMCU_{\mathrm{OI}}$. This means that applying a function decoder drastically reduces the total number of required logic elements.

On the other hand, the number of flip-flops and dedicated memory blocks required to realise a controller in the form of a $CMCU_{\mathrm{OI}}$ and especially $CMCU_{\mathrm{OD}}$ is greater than in the case of a $CMCU_{\mathrm{MM}}$ and an FSM. As expected, applying a function decoder resulted in fewer logic elements (LUTs and slices) but increased the number of BRAMs. The above analysis may be summarised as follows:

- For systems not containing many elements that should be implemented with dedicated memories, the *CMCU with identification of outputs and a function decoder* should be selected for controller implementation. Such realisations permit to reduce the number of logic blocks by over 40% in comparison with CMCUs with mutual memory but require more (at least one) dedicated memory blocks for implementation.

- For systems that may use wide areas of dedicated memory blocks, the *CMCU with identification of outputs* appears to be the best solution. Although it consumes more BRAMs than a controller with mutual memory (the difference is about 2%), this approach reduces the total number of logic elements on the average by about 24%. Moreover, such a realisation consumes much fewer BRAMs than a microprogrammed controller with function decoder does.

6. Conclusion

Methods to design microprogrammed controllers were proposed and discussed. The method of designing CMCUs with mutual memory was improved by modifying the structure of the control units. In CMCUs with identification of outputs, microinstruction addresses were encoded in a special way, yielding a reduction in the number of internal variables required to recognise the current controller state. The method was further improved by employing function decoders, which may be implemented with dedicated memory blocks of the target FPGAs. The presented methods aim at reducing the number of logic elements required in programmable devices. The results of experiments show that the best gain is achieved by implementing controllers as CMCUs with function decoders. However, such a realisation consumes more dedicated memories than other CMCUs. Therefore, if other modules of a system require many BRAM resources, microprogrammed controllers with identification of outputs are the best solution.

References

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

Baranov, S. I. (1994). *Logic Synthesis for Control Automata*, Kluwer Academic Publishers, Boston, MA.

Barkalov, A. and Titarenko, L. (2009). *Logic Synthesis for FSM-Based Control Units*, Springer-Verlag, Berlin.

Barkalov, A., Węgrzyn, M. and Wiśniewski, R. (2006). Optimization of LUT-elements amount in cotrol unit of system-on-chip, *Discrete-Event System Design, DESDes '06: A Proceedings Volume from the 3rd IFAC Workshop, Rydzyna, Poland*, pp. 143–146.

Barkalov A., Titarenko L. andChmielewski S. (2007). Reduction in the number of PAL macrocells in the circuit of a Moore FSM, *International Journal of Applied Mathematics and Computer Science* **17**(4): 565–675, DOI: 10.2478/v10006-007-0046-8.

Brown, S. and Vranesic, Z. (2000). *Fundamentals of Digital Logic with VHDL Design*, McGraw Hill, New York, NY.

Bukowiec, A. (2009). *Synthesis of Finite State Machines for FPGA Devices Based on Architectural Decomposition*, University of Zielona Góra Press, Zielona Góra.

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

Gajski, D. (1996). *Principles of Digital Design*, Prentice Hall, Upper Saddle River, NJ.

Garcia-Vargas, I., Senhadji-Navarro, R., Jimenez-Moreno, G., Civit-Balcells, A. and Guerra-Gutierrez, P. (2007). ROM-based finite state machine implementation in low cost FPGAs, *IEEE International Symposium on Industrial Electronics (ISIE), Vigo, Spain*, pp. 2342–2347.

Kania, D. (2004). *The Logic Synthesis for the PAL-based Complex Programmable Logic Devices*, Lecture Notes of the Silesian University of Technology, Gliwice, (in Polish).

Lee, J.M. (1999). *VERILOG QuickStart: A Practical Guide to Simulation and Synthesis in VERILOG*, Kluwer Academic Publishers, Norwell, MA.

Łuba, T. (2005). *Synthesis of Logic Devices*, Warsaw University of Technology Press, Warsaw, (in Polish).

Łuba, T., Borowik, G. and Kraśniewski, A. (2009). Synthesis of finite state machines for implementation with programmable structures, *Electronics and Telecommunications Quarterly* **55**(2): 183–200.

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

Sentovich, E.M. (1993). *Sequential Circuit Synthesis at the Gate Level*, Ph.D. thesis, University of California, Berkeley, CA.

Thomas, D. and Moorby, P. (2002). *The Verilog Hardware Description Language*, 5th Edn., Kluwer Academic Publishers, Norwell, MA.

Wiśniewska, M., Wiśniewski, R. and Adamski, M. (2007). Usage of hypergraph theory in decomposition of concurrent automata, *Pomiary, Automatyka, Kontrola* (7): 66–68.

Wiśniewski, R. (2005). Partial reconfigutration of microprogrammed controllers implemented in FPGAs, *Proceedings of the International Ph.D. Workshop OWD 2005, Wisła, Poland*, Vol. 21, pp. 239–242, (in Polish).

Wiśniewski, R. (2009). *Synthesis of Compositional Microprogram Control Units for Programmable Devices*, University of Zielona Góra Press, Zielona Góra.

Wiśniewski, R., Barkalov, A. and Titarenko, L. (2006). Optimization of address circuit of compositional microprogram unit, *Proceedings of the IEEE East-West Design & Test Workshop, EWDTW '06, Sochi, Russia*, pp. 167–170.

Zwolinski, M. (2000). *Digital System Design with VHDL*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Remigiusz Wiśniewski was born in Poznań in 1978. He received his Ph.D. degree in computer science from the University of Zielona Góra in 2008. His general research interests include cryptology, design and verification methods for digital circuits and application of programming languages to computer-aided design systems. Since 2003 he has been an assistant professor at the Institute of Computer Engineering and Electronics of the University of Zielona Góra.

Alexander Barkalov was born in Russia in 1954. Since 2003 he has been a professor of computer engineering at the Institute of Computer Engineering and Electronics, University of Zielona Góra, Poland, and he is also a professor at the Institute of Computers, Donetsk National Technical University, Ukraine. His current research interests include the theory of digital automata, especially methods for synthesis and optimization of control units implemented with field-programmable logic devices. He has taken part in a number of research projects sponsored by different institutions of the former USSR. These projects involved the development of computer-aided design tools to implement control units. He has published more than 400 papers in international journals and conference proceedings, and is the author of two and a co-author of eight monographs.

Larisa Titarenko was born in Ukraine in 1971. Since 2007 she has been a professor of telecommunications at the Institute of Computer Engineering and Electronics, University of Zielona Góra, Poland, and she is also a professor at the Institute of Telecommunications Systems, Kharkov National University of Radioelectronics, Ukraine. Her current research interests include telecommunication systems, antennas and digital automata, and their of applications. She has taken part in a number of research projects sponsored by the Ministry of Science and Higher Education of Ukraine between 1993 and 2005. She has published more than 120 papers in international journals and conference proceedings, and is the author or a co-author of four monographs.

Wolfgang A. Halang was born in Essen, Germany, in 1951. He received doctoral degrees in mathematics and computer science from Bochum and Dortmund Universities, respectively, worked both in industry (Coca-Cola GmbH and Bayer AG) and academia (University of Petroleum and Minerals, Saudi Arabia, and University of Illinois at Urbana-Champaign, USA), before he was appointed to the Chair of Applications-Oriented Computing Science and the head of the Department of Computing Science at the University of Groningen in the Netherlands. Since 1992 he has been holding the Chair of Computer Engineering at the Faculty of Electrical and Computer Engineering at Fernuniversität in Hagen, Germany. His research interests comprise hard real-time computing with special emphasis on safety-related systems. He is the founder and was the European editor-in-chief of the journal *Real-Time Systems*, a member of the editorial boards of four other journals, a co-director of the 1992 NATO Advanced Study Institute on Real-Time Computing. He has authored 10 books and some 350 refereed book chapters, journal publications and conference contributions, edited 16 books, and holds 12 patents.

Received: 10 May 2010
Revised: 1 September 2010