DESIGN OF PSEUDO-EQUIVALENT MICROPROGRAM AUTOMATA ON PROGRAMMABLE LOGIC DEVICES

Irena Bulatowa, Mateusz Radziwoniuk

Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland

Abstract: In this paper, a new method of synthesis of microprogram automata from ASM specification is presented. This method allows converting pseudo-equivalent automaton to an equivalent one by eliminating the zero-value output sets appearing in additional internal states. The proposed method is based on a modified model of microprogram automaton, which permits changing the output signals only in the basic internal states, thereby eliminating the zero-value sets of output signals generated in additional states of pseudo-equivalent automata. This allows removing the adverse effects of introducing additional states and provides a wider application of numerous methods for the synthesis of pseudo-equivalent microprogram automata. The experimental results show that the cost of realization of the proposed structure in programmable logic devices increases insignificantly, but then it leads to extend the field of application synthesis methods based on the introduction of additional internal states.

Keywords: microprogram automaton, Algorithmic State Machine (ASM), pseudo-equivalent automaton, additional internal states, programmable logic devices (PLD)

1. Introduction

Developing effective methods for synthesis of microprogram automata on programmable logic devices (PLD) is a very important problem because the majority of control systems are based on the principle of microprogram control [7]. The behavior of microprogram automata is very often specified by Algorithmic State Machine (ASM) charts [2], which are very useful and convenient methods of control algorithm description. Lots of methods for the synthesis of microprogram automata from ASM specifications have been developed [2,3,5,1,6]. Some of these synthesis methods require the introduction of additional internal states for receiving special features of designed microprogam automata. For example, some methods based on additional internal states may allow simplifying the microprogram automata scheme

Zeszyty Naukowe Politechniki Białostockiej. Informatyka, vol. 7, pp. 17-29, 2011.

and reducing the cost of their realization [1], and other methods [1,6] allow including restrictions on the number of inputs of components used for microprogram automata realization.

However, the introduction of additional internal states is not always acceptable in microprogram automata design. This is due to the fact that the zero-value output sets are generated in additional internal states. It may result in damage to the functioning of the designed microprogram automata when the output signals must be maintained at a constant high level and any changes of the signal level are not permitted.

As a result of the introduction of additional internal states during the synthesis process, the pseudo-equivalent automaton will be received. Pseudo-equivalent automaton generates the same sequence of output signals as original automaton, but differs from it in that the zero-value output sets are generated in the output sequence of pseudo-equivalent automaton in additional states. That fact significantly reduces the application area of synthesis methods based on the introduction of additional internal states.

In this paper, a new method for the synthesis of microprogram automata from ASM specification is presented. This method allows converting the pseudo-equivalent automaton into an equivalent one by eliminating the zero-value output vectors appearing in additional internal states. The proposed method is based on a modified model of microprogram automaton, which allows triggering the output signals only in basic internal states thereby eliminating the zero-value sets on automaton outputs. The generation of an additional control signal in the proposed model leads to a slight increase in the complexity of automaton realization, but then it makes it possible to apply numerous methods for the synthesis of pseudo-equivalent microprogram automata, even in such applications in which it was previously impossible.

2. Synthesis of microprogram automata from ASM

Due to the principle of microprogram control [7], any complex operation executed by a digital device is represented as a sequence of elementary operations $Y = \{y_1, \ldots, y_N\}$, called *microoperations*. The subset $Y^t \subseteq Y$ of microoperations executed in the same clock period forms a *microinstruction*. The order of microinstructions execution is determined by logical conditions $X = \{x_1, \ldots, x_L\}$. The control algorithm specified in terms of microoperations and logical conditions is called a *microprogram* and the automaton, which realizes the microprogram, is called a *microprogram automaton* [6].

The Algorithmic State Machine (ASM) charts [2] (Fig.1) are widely used for control algorithm specification. Each operator vertex of ASM contains a microin-

struction $Y^t \subseteq Y$, $Y = \{y_1, \ldots, y_N\}$ defined as a collection of microoperations which are executed in the same clock period, and $Y^t = \varnothing$ is acceptable. One of the logical conditions from the set $X = \{x_1, \ldots, x_L\}$ is written in each conditional vertex and it is possible to write the same logical condition in different vertices [2].

The finite state machine (FSM) is used as a model of microprogram automata. Synthesis of FSM from the ASM chart begins from the construction of marked ASM, due to which the ASM chart is marked by labels $A = \{a_1, \ldots, a_M\}$ corresponding to internal states of FSM [2]. Standard approaches to Moore and Mealy FSM synthesis are well known [2,3]. Labels and corresponding internal states introduced by these standard algorithms will be called *basic labels* and *basic* internal states.

Due to the standard algorithm for the synthesis of Mealy FSM from ASM chart [3], the input vertex following the initial vertex *Begin* and the input of vertex *End* are marked by the symbol a_1 (corresponding to the initial state of automaton), then the inputs of vertices following operator vertices are marked by symbols $a_2, \ldots, a_M$. This algorithm of ASM marking allows building FSM in which the output functions $y_1, \ldots, y_N$ will depend on the current internal states and input variables $x_1, \ldots, x_L$.

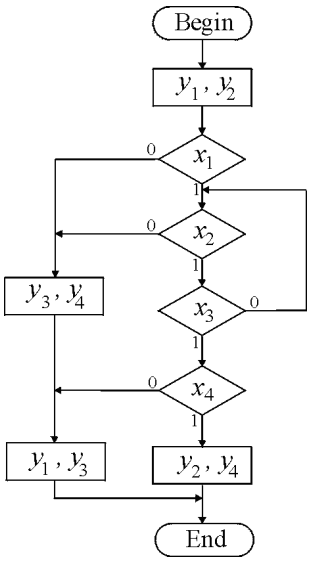

Fig. 1. An example of ASM chart

A graph (or transition table) of automaton is constructed from the marked ASM by defining all the transition paths between internal states: $a_m\, X(a_m, a_s)\, Y(a_m, a_s)\, a_s$,

where $X(a_m, a_s)$ – the product of logical conditions on the transition path from a_m to a_s, $a_m, a_s \in A$; $Y(a_m, a_s)$ – microinstruction generated on this transition.

For Moore automaton synthesis, the marked ASM is constructed as follows [2]: vertices *Begin* and *End* are marked by the same symbol a_1, and all operator vertices are marked by different symbols $a_2, \ldots, a_M$. This algorithm allows implementing the FSM with output functions $y_1, \ldots, y_N$ depending only on the current state of the automaton.

Many methods for FSM synthesis from ASM have been developed in which the additional internal states are introduced besides the basic internal states. In such methods, the additional symbols $a_{M+1}, \ldots, a_{M+K}$ are used for marking ASM that leads to the introduction of additional internal states of FSM.

Increasing the number of internal states allows the automata to acquire new properties. For example, in the synthesis method proposed in [1], additional states are used for minimizing the number of transitions between states that can reduce the complexity of automata realization.

In methods [1,6], the additional states are introduced to decrease the dependency of the transition and output functions on input variables. In these algorithms, after the standard marking of ASM, the additional labels $a_{M+1}, \ldots, a_{M+K}$ are placed at the inputs of conditional vertices. This makes it possible to reduce the rank of the conjunctions $X(a_m, a_s)$ defining the automaton transitions, which results in reducing the rank of the products in transition functions $d_1, \ldots, d_R$ and in output functions $y_1, \ldots, y_N$. It may be important if there are restrictions on the number of inputs of components used for FSM realization.

As a result of the introduction of additional states, the pseudo-equivalent automaton will be obtained [6]. Let $z_1, \ldots, z_k$ be some sequence of input variables vectors on automaton inputs, and $w_1, \ldots, w_k$ will be the corresponding sequence output vectors generated on automaton outputs. Two automata S_1 and S_2 are called *equivalent*, if they generate the same output sequences $w_1, \ldots, w_k$ for each input sequence $z_1, \ldots, z_k$. An automaton S_2 is called *pseudo-equivalent* to automaton S_1, if it generates the same output sequence as S_1, but in its output sequence zero-value output vectors may appear as a result of the introduction of additional states [6].

Zero-value output vectors correspond to paths in ASM which don't pass through an operator vertex (for Mealy automaton) or to paths which don't lead to an operator vertex (for Moore automaton). Such paths end in some additional label a_j, $j > M$. The appearance of zero-value vectors may be unacceptable in some applications when it is important to maintain the output signals at a constant high level.

Let us consider an example of ASM shown in Fig.2. At the beginning, the ASM has been marked by symbols $a_1, \ldots, a_5$ according to the standard algorithm for the

synthesis of Moore FSM [2]. Then, the additional labels a_6, a_7 and a_8 have been introduced to reduce the dependency of FSM transition functions from input variables [1]. According to this algorithm, the inputs of conditional vertices connected by the edge with the output of other conditional vertex are marked by additional labels, which leads to the introduction of additional internal states. As a result, each transition path of FSM will depend on no more than one logical condition, which allows limiting the maximum rank of conjunctions in transition functions of Moore FSM. This can be useful when there are hard restrictions on the number of inputs of elements used for circuit realization.

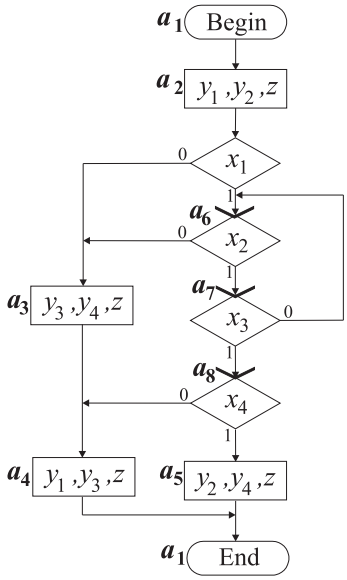

Fig. 2. Marked ASM for synthesis of Moore FSM with additional labels

However, the introduction of additional internal states may be unacceptable in some applications. Let us consider the transition path between basic states a_2 and a_5 on ASM presented in Fig.2. In both states, the output signal y_2 is generated. If in practical application, it is required to maintain the signal y_2 at a constant high level on the transition from a_2 to a_5, the introduction of additional states between a_2 and a_5 will be unacceptable, because in additional states a_6, a_7 and a_8 signal y_2 will be temporarily triggered to a low level. A similar situation is also possible for output signal y_1 on transition from a_2 to a_4 (Fig.2). This fact narrows the field

of application of synthesis methods based on the introduction of additional internal states and requires a preliminary inspection of the control algorithms before applying such synthesis methods.

In this paper, we propose a modified model of microprogram automaton, which allows eliminating the zero-value output sets in additional internal states and gives the ability of a wider application of synthesis methods of pseudo-equivalent automata.

3. Modified model of microprogram automaton

The modified model of microprogram automata is presented in Fig.3. The register RG stories the current automaton state code $e_1, \ldots, e_R$, where $R =]log_2 M[$ is the least integer greater than or equal to $log_2 M$. The combinational logic circuit CL implements the output functions $y_1, \ldots, y_N$ and the transition functions $d_1, \ldots, d_R$.

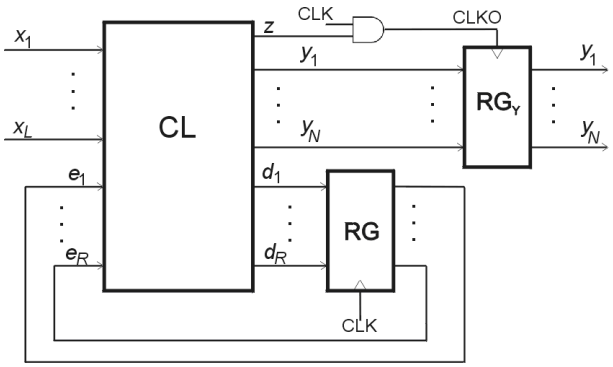

Fig. 3. Modified structure of microprogram automata

An additional register RG_Y is introduced in this model to store the output functions values $y_1, \ldots, y_N$. The special pulse CLKO is used to change the content of the RG_Y register. The signal CLKO is generated on the basis of clock signal z formed by combinational circuit CL (Fig.4). The waveforms for CLKO signal generation are shown in Fig.4, where the clock periods corresponding to the basic internal states are marked by arrows. An additional signal $z = 1$ is formed by combinational logic circuit CL only in the basic internal states, but in additional states $z = 0$.

This causes that the content of register RG_Y will be changed only in basic internal states, and the zero-value output vectors generated in the additional states will not be written to register RG_Y, so they never appear on the automaton outputs $y_1, \ldots, y_N$.

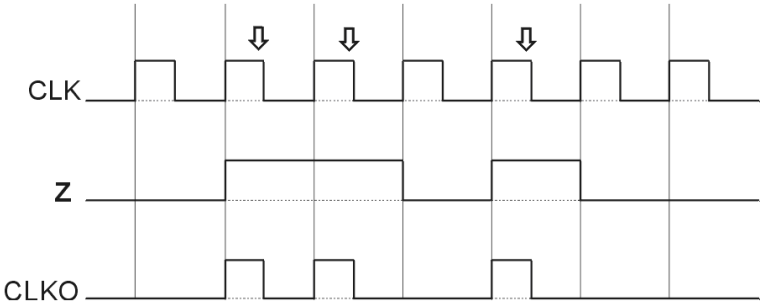

Fig. 4. Waveforms for CLKO signal forming

The implementation of additional register RG_Y in PLD structures does not increase the number of used macrocells for FSM realization, since the internal memory elements of macrocells are used for its implementation. The complexity of realization of the proposed structure has increased insignificantly; it is related to the implementation of only one additional function z.

To implement function z, all microinstructions generated in basic internal states should be expanded by one extra microoperation z that will be correspond to forming the microoperation signal $z = 1$ only in the basic internal states.

In our example (Fig.3), the ASM was marked for synthesis of Moore FSM and then the additional states a_6, a_7 and a_8 were introduced. According to the proposed synthesis method, an additional microoperation z must be inserted in all operator vertices of ASM, because each operator vertex corresponds to the basic state of Moore FSM.

The structure table of Moore automaton with additional microoperation z is shown in Table 1, where each transition is described by the following columns: a_m is the current FSM state, $K(a_m)$ is the code of the state a_m, a_s is the next state, $K(a_s)$ is the code of the state a_s, $X(a_m, a_s)$ is the conjunction of inputs determining the transition, $Y(a_m)$ is the microinstruction generated in the state a_m, $D(a_m, a_s)$ is a collection of transition functions for D-type memory elements.

On the basis of the structure table, the following expressions for output functions $y_1, \ldots, y_4$, transition functions $d_1, \ldots, d_3$ and for additional function z are obtained:

$y_1 = \overline{e}_1\overline{e}_2e_3 + \overline{e}_1e_2e_3$
$y_2 = \overline{e}_1\overline{e}_2e_3 + e_1\overline{e}_2\overline{e}_3$
$y_3 = \overline{e}_1e_2\overline{e}_3 + \overline{e}_1e_2e_3$
$y_4 = \overline{e}_1e_2\overline{e}_3 + e_1\overline{e}_2\overline{e}_3$

Table 1. Structure table of automaton

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m,a_s)$	$Y(a_m)$	$D(a_m,a_s)$
a_1	000	a_2	001	1	–	d_3
a_2	001	a_3	010	$\overline{x}_1$	y_1, y_2, z	d_2
		a_6	101	x_1		$d_1\ d_3$
a_3	010	a_4	011	1	y_3, y_4, z	$d_2\ d_3$
a_4	011	a_1	000	1	y_1, y_3, z	d_3
a_5	100	a_1	000	1	y_2, y_4, z	d_3
a_6	101	a_3	010	$\overline{x}_2$	–	d_2
		a_7	110	x_2		$d_1\ d_2$
a_7	110	a_6	101	$\overline{x}_3$	–	$d_1\ d_3$
		a_8	111	x_3		$d_1\ d_2\ d_3$
a_8	111	a_4	011	$\overline{x}_4$	–	$d_2\ d_3$
		a_5	100	x_4		d_1

$z = \overline{e}_1 e_2 \overline{e}_3 + \overline{e}_1 e_2 \overline{e}_3 + \overline{e}_1 e_2 e_3 + e_1 \overline{e}_2 \overline{e}_3$
$d_1 = \overline{e}_1 \overline{e}_2 e_3 x_1 + e_1 \overline{e}_2 e_3 x_2 + e_1 e_2 \overline{e}_3 \overline{x}_3 + e_1 e_2 \overline{e}_3 x_3 + e_1 e_2 e_3 x_4$
$d_2 = \overline{e}_1 \overline{e}_2 e_3 \overline{x}_1 + \overline{e}_1 e_2 \overline{e}_3 + e_1 \overline{e}_2 e_3 \overline{x}_2 + e_1 \overline{e}_2 e_3 x_2 + e_1 e_2 \overline{e}_3 x_3 + e_1 e_2 e_3 \overline{x}_4$
$d_3 = \overline{e}_1 \overline{e}_2 \overline{e}_3 + \overline{e}_1 \overline{e}_2 e_3 x_1 + \overline{e}_1 e_2 \overline{e}_3 + e_1 e_2 \overline{e}_3 \overline{x}_3 + e_1 e_2 \overline{e}_3 x_3 + e_1 e_2 e_3 \overline{x}_4$

In our example, the complexity of FSM realization has increased slightly due to forming of an additional function z, which contains the same products as output functions of Moore FSM.

The zero-value output sets in additional internal states could also be eliminated in another way by simple repeating in additional states of the microoperations that are generated in the preceding basic state. However, such an approach leads to a significant increase in the complexity of output functions realization, so it will be less effective compared with the proposed method.

4. Experimental results

The proposed method was tested using the control algorithms from the ASM library of the Abelite EDA tool [4]. To perform the experiments, three methods using additional states have been implemented and compared: M1 – the method, in which the zero-value output vectors appear in additional states [1]; M2 – the proposed method, in which the additional signal z is formed to eliminate the zero-value output sets; M3 – the method, in which the zero-value output sets are eliminated by the repeating in additional states of the microoperations from the previous basic state. All these methods introduce the same number of additional states to separate all conditional

vertices on the ASM chart. For all the methods, the number of used macrocells of PLD were compared for the realization of automata on FLEX10K and MAX9000 devices of Altera, which are the typical representatives of two PLD classes: CPLD (MAX9000 device family) and FPGA (FLEX10K devices).

Table 2 shows the results of comparison of methods M1, M2 and M3 for the FLEX10K device family, where "ASM" is the name of the example from the ASM library, L, N, S, K are the numbers of inputs, outputs, states and additional states of automaton, respectively, C_{M1}, C_{M2}, C_{M3} are the numbers of macrocells of the FLEX10K device used for the realization automata for synthesis methods M1, M2 and M3, respectively. For methods M2 and M3, the values P_{M2} and P_{M3} have been calculated as: $P_{M2} = \frac{C_{M2}-C_{M1}}{C_{M1}}$, $P_{M3} = \frac{C_{M3}-C_{M1}}{C_{M1}}$, where P_{M2} and P_{M3} are the percent of growth of the number of used macrocells for methods M2 and M3, respectively, in comparison with method M1.

Analysis of the obtained results show that the number of used macrocells for the proposed method M2 increases on average 3.65% (1.38% in the best case) in comparison with method M1. The method M3, as it was expected, requires a significantly greater increase in the amount of hardware, on average 23.75% (even 38.19% in the worst case), so method M3 is much less effective in comparison with the proposed method M2.

The results presented in Table 3 show the dependency of value PM2 on such a parameter as the percentage of conditional blocks in ASM (B_X/B), where B is the whole number of blocks in ASM, B_X is the number of conditional blocks, L, N, S are the numbers of inputs, outputs, and states of automaton, respectively.

The results from Table 3, which show the correlation between the percentage of conditional block in ASM (B_X/B) and the growth of the number of macrocells used for FSM realization by method M2, are also presented in a scatter graph in Fig.5. The pattern of dots suggests the falling correlation between the parameters, thus for the tested examples the complexity growth rate for method M2 reduces with the increase of the percentage of conditional blocks in ASM.

Table 4 shows the results of comparison of methods M1, M2 and M3 for realization of automata on MAX9000 devices of Altera.

Analysis of the results presented in Table 4 shows that the number of used macrocells for the proposed method M2 increases on average 3.43% (0.55% in the best case) in comparison with method M1. And in the case of method M3, the complexity of realization increases significantly, on average 17.95% (even 39.78% in the worst case), so the proposed method M2 is a more effective approach to eliminating zero-value output sets.

Table 2. Comparison of synthesis methods for realization on FLEX 10K device family

ASM	L	N	S	K	C_{M1}	C_{M2}	C_{M3}	P_{M2}	P_{M3}
acdl	16	27	174	151	330	335	398	1.52%	20.61%
araf	25	65	124	48	230	240	279	4.35%	21.30%
ass13	5	25	38	19	89	91	110	2.25%	23.60%
berg	21	51	121	51	224	234	285	4.46%	27.23%
cpu	14	29	44	21	83	87	96	4.82%	15.66%
cyr	20	75	132	55	244	258	292	5.74%	19.67%
e1	12	13	114	90	201	205	274	1.99%	36.32%
e6	11	20	41	23	82	86	93	4.88%	13.41%
e15	13	20	85	67	163	167	198	2.45%	21.47%
klain	27	61	134	55	251	258	300	2.79%	19.52%
kobz	19	53	130	59	235	245	293	4.26%	24.68%
lcu	15	24	81	58	144	150	199	4.17%	**38.19%**
lior	24	31	116	79	188	198	255	5.32%	35.64%
max	26	41	105	56	189	197	233	4.23%	23.28%
micks	21	45	106	53	195	200	254	2.56%	30.26%
pilot	27	22	60	33	111	115	138	3.60%	24.32%
raz	23	72	131	60	238	248	300	4.20%	26.05%
sasi	19	54	129	54	240	251	293	4.58%	22.08%
structm	33	36	106	87	217	220	234	**1.38%**	7.83%
v16	14	18	89	72	164	167	211	1.83%	28.66%
oshr	19	72	144	53	257	266	307	3.50%	19.46%
e16	13	18	85	67	161	166	190	3.11%	18.01%
e8	13	20	85	67	163	167	198	2.45%	21.47%
bcomp	18	39	68	33	122	129	145	5.74%	18.85%
asm1	15	22	32	19	58	61	79	5.17%	36.21%
Average								**3.65%**	**23.75%**

5. Conclusions

The proposed method can be used for eliminating zero-value output vectors, which appear in additional internal states of microprogram automata. This method is based on a modified model of microprogram automata, which allows removing the adverse effects of introducing additional states in exchange for a slight increase in the amount of hardware. The experimental results show that the proposed method is more effective (on average 15.37% for realization on FLEX 10K devices and 19.11% for realization on MAX9000 devices) than the approach based on repeating output signals in additional states.

Table 3. Dependency of synthesis results on parameters of ASM and FSM

ASM	L	N	S	B	B_X	B_X/B	C_{M1}	C_{M2}	P_{M2}
acdl	16	27	174	194	171	88.14%	330	335	1.52%
alf	31	74	127	160	83	51.88%	241	262	8.71%
araf	25	65	124	134	58	43.28%	230	240	4.35%
ass13	5	25	38	52	33	63.46%	89	91	2.25%
bech	18	39	68	72	37	51.39%	119	128	7.56%
berg	21	51	121	132	62	46.97%	224	234	4.46%
bs	19	13	127	144	127	88.19%	214	219	2.34%
cat	11	22	30	37	20	54.05%	58	62	6.90%
cpu	14	29	44	49	26	53.06%	83	87	4.82%
cyr	20	75	132	140	63	45.00%	244	258	5.74%
e1	12	13	114	135	111	82.22%	201	205	1.99%
e6	11	20	41	50	32	64.00%	82	86	4.88%
e15	13	20	85	102	84	82.35%	163	167	2.45%
klain	27	61	134	153	74	48.37%	251	258	2.79%
kobz	19	53	130	141	70	49.65%	235	245	4.26%
lcu	15	24	81	99	76	76.77%	144	150	4.19%
lift	14	30	42	56	24	42.86%	79	86	8.86%
lior	24	31	116	134	97	72.39%	188	198	5.32%
max	26	41	105	113	64	56.64%	189	197	4.23%
micks	21	45	106	115	62	53.91%	195	200	2.56%
pilot	27	22	60	70	43	61.43%	111	115	3.60%

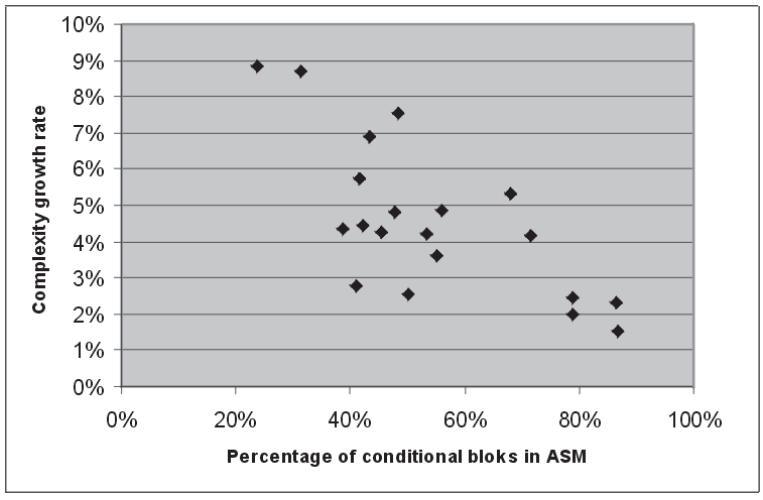

Fig. 5. Scatter graph for complexity growth rate in relation to percentage of conditional blocks

Table 4. Comparison of synthesis methods for realization on MAX9000 device family

ASM	L	N	S	K	C_{M1}	C_{M2}	C_{M3}	P_{M2}	P_{M3}
alf	31	74	127	50	146	153	166	4.79%	13.70%
araf	25	65	124	48	140	152	164	8.57%	17.14%
ass13	5	25	38	19	54	55	61	1.85%	12.96%
bech	18	39	68	33	69	73	74	5.80%	7.25%
berg	21	51	121	51	133	138	171	3.76%	28.57%
big	18	28	127	110	182	183	189	**0.55%**	3.85%
bs	19	13	127	110	165	168	184	1.82%	11.52%
e1	12	13	114	90	166	167	196	0.60%	18.07%
e15	13	20	85	67	110	111	125	0.91%	13.64%
klain	27	61	134	55	158	163	184	3.16%	16.46%
kobz	19	53	130	59	130	138	163	6.15%	25.38%
lcu	15	24	81	58	93	95	130	2.15%	**39.78%**
lift	14	30	42	10	50	51	53	2.00%	6.00%
max	26	41	105	56	128	130	151	1.56%	17.97%
micks	21	45	106	53	114	121	138	6.14%	21.05%
pp	20	28	89	72	120	121	137	0.83%	14.17%
pilot	27	22	60	33	62	67	82	8.06%	32.26%
sasi	19	54	129	54	136	144	175	5.88%	28.68%
oshr	19	72	144	53	150	154	174	2.67%	16.00%
e8	13	20	85	67	107	108	125	0.93%	16.82%
bcomp	18	39	68	33	73	77	86	5.48%	17.81%
asm1	15	22	32	19	38	39	47	2.63%	23.68%
asm2	15	22	31	17	40	41	44	2.50%	10.00%
Average								**3.43%**	**17.95%**

References

[1] Baranov S., Sklarov V., Digital systems based on programmable circuits with matrix structure, Moscow: Radio i sviaz, 1986 (in Russian).
[2] Baranov S., Logic Synthesis for Control Automata, Kluwer Academic Publishers, 1994.
[3] Baranov S., Logic and System Design of Digital Systems, Tallinn: TTU Press and SiB Publishers, 2008.
[4] Baranov S., High level synthesis in EDA tool "Abelite", Electronics and Telecommunications Quarterly, 2009, Vol.55, No.2, pp. 123-156.
[5] Barkalov A., Titarenko L., Logic synthesis for compositional microprogram control units, Berlin: Springer-Verlag, 2008.
[6] Salauyou V., Klimowicz A., Logical synthesis of digital devices in PLD structures, Bialystok: OWPB, 2010 (in Polish).
[7] Wilkes M.V., The Genesis of Microprogramming, IEEE Annals of the History of Computing, 1986, V.8, No.2, pp.116-126.

PROJEKTOWANIE PSEUDOEKWIWALENTNYCH AUTOMATÓW MIKROPROGRAMOWALNYCH NA UKŁADACH PLD

Streszczenie Metody syntezy automatów mikroprogramowalnych oparte na wprowadzeniu dodatkowych stanów wewnętrznych prowadzą do otrzymania automatów pseudoekwiwalentnych. Sekwencja słów wyjściowych takich automatów naruszana jest pojawieniem się zerowych słów wyjściowych w stanach dodatkowych, co nie zawsze jest dopuszczalne w zastosowaniach praktycznych. W artykule została przedstawiona nowa metoda syntezy automatów mikroprogramowalnych, która pozwala przekształcić automat pseudoekwiwalentny na postać ekwiwalentną. Zaproponowana została zmodyfikowana struktura automatu mikroprogramowalnego, w której zmiana sygnałów wyjściowych jest możliwa wyłącznie w stanach podstawowych, tym samym eliminuje się słowa zerowe na wyjściach automatu. Badania eksperymentalne pokazały, że złożoność realizacji zaproponowanej struktury na układach programowalnych wzrasta w nieznacznym stopniu, natomiast takie podejście pozwala znacznie rozszerzyć obszar zastosowania metod syntezy automatów mikroprogramowalnych opartych na wprowadzeniu dodatkowych stanów wewnętrznych.

Słowa kluczowe: automat mikroprogramowalny, sieć działań, automat pseudoekwiwalentny, dodatkowe stany wewnętrzne, programowalne układy logiczne.