BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 54, No. 4, 2006

FSMs state encoding targeting at logic level minimization

R. CZERWIŃSKI*, D. KANIA, and J. KULISZ

Institute of Electronics, Silesian University of Technology, 16 Akademicka Str., 44-100 Gliwice, Poland

Abstract. The paper concerns the problem of state assignment for finite state machines (FSM), targeting at PAL-based CPLDs implementations. Presented in the paper approach is dedicated to state encoding of fast automata. The main idea is to determine the number of logic levels of the transition function before the state encoding process, and keep the constraints during the process. The number of implicants of every single transition function must be known while assigning states, so elements of two level minimization based on Primary and Secondary Merging Conditions are implemented in the algorithm. The method is based on code length extraction if necessary. In one of the most basic stages of the logic synthesis of sequential devices, the elements referring to constraints of PAL-based CPLDs are taken into account.

Key words: state assignment, finite state machines (FSM), programmable array logic (PAL), complex programmable logic devices (CPLD).

1. Introduction

Proper state encoding is very important for sequential logic. The choice of the codding words assigned to states of an FSM has a tremendous influence on the number of flip-flops and complexity of the transition and output combinatorial blocks. The sequential automata state assignment plays therefore a major role in the synthesis process.

The main problem of the synthesis is implementation of a single-output function, which is a sum of p-implicants, by means of logic cells containing k-terms, if $p > k$. The problem appears when targeting PAL-based CPLDs implementations. The implementation of such a function requires more than one cell. This way, the number of logic levels of such implementation is increased, and so propagation time is increased. The methods of avoiding this problem in combinatorial devices has been studied for instance in [1–4].

There are many different methods of state assignment. This is because there are different goals of optimization and many structures of automata. Some well known methods, e.g. "one hot" coding, are simple, but they often give effects far from optimum. There are also methods which give results considered as optimal [5–7]. The state assignment problem is often solved together with input and output encoding [8]. Some methods are based on dichotomies or dominance graphs [6,9]. Sometimes the problem is solved using genetic algorithms [10]. The most importand and popular academic systems are: NOVA [6], MUSTANG [11], JEDI [12], ASYL [1,2,13].

A large majority of methods are dedicated for automata which are to be implemented in a PLA-based devices, while the kernel of most CPLDs is a PAL-based cell. A characteristic feature of this cell is a limited number of terms connected to the OR-gate. The output of an AND-gate is connected to only one OR-gate. So, the product terms cannot be shared among the functions, and unused terms cannot be freely allocated to other cells.

The aim of the proposed state assignment method is to fit the finite state machines to the structure of the PAL-based CPLDs as well as possible. The major purpose is taking into account – minimizing the number of logic level of the transition function. The number of logic levels of the transition function is determined before the state assignment process on the basis of state weights. Of course, minimization of the number of PAL-based cells used to implement the transition function is also taken into account. Elements of two-level minimization, based on Primary and Secondary Merging Conditions, are included in the algorithm. The main idea is to extract the length of the coding word if necessary (when the number of logic levels exceeds the assumed value).

The method, dedicated for fast automata, is quick and easy. Because of a heuristic nature of the algorithm, many experiments were carried out, and are reported in the paper.

This paper is structured as follows. Section 2 introduces some basic informations about PAL-cell and automata theory. Section 3 focuses on the basic ideas and definitions. The algorithms are presented in Section 4. Experimental results are reported in Section 5. The paper concludes with a summary in Section 6.

2. Preliminaries

2.1. The automata theory. Let the sequential circuit has n-inputs, m-outputs, and K-memory elements to represent l-internal states of the circuit. The input state is determined by the vector $X = (x_{n-1}, \ldots, x_0)$, the output state is determined by the vector $Y = (y_{m-1}, \ldots, y_0)$, and the internal state is determined by the vector $S = (Q_{K-1}, \ldots, Q_0)$. Let $\mathbb{B} = \{0, 1\}$.

The mathematical model of a sequential circuit is a Finite State Machine (FSM), which is a six-tuple: $\{\mathbb{X}, \mathbb{Y}, \mathbb{S}, \delta, \lambda, S_r\}$, where: $\mathbb{X}$ is a finite input alphabet ($X \in \mathbb{X} \subseteq \mathbb{B}^n$), $\mathbb{Y}$ is a finite output alphabet ($Y \in \mathbb{Y} \subseteq \mathbb{B}^m$), $\mathbb{S}$ is a finite set of states ($S \in \mathbb{S} \subseteq \mathbb{B}^l$), $\delta\colon \mathbb{X} \times \mathbb{S} \to \mathbb{S}$ is the transition function, λ is the output function and $S_R \in \mathbb{S}$ is the initial or reset state.

The transition function of an FSM determines next state

*e-mail: robert.czerwinski@polsl.pl

of the automata (S^+), and is the mapping $\delta\colon \mathbb{X} \times \mathbb{S} \rightarrow \mathbb{S}$ ($S^+ = \delta(X, S)$).

The most popular finite state machines are the Mealy FSM, and the Moore FSM. In the Mealy FSM the output function is associated with each transition: $\lambda\colon \mathbb{X}\times\mathbb{S} \rightarrow \mathbb{Y}$ ($Y = \lambda(X, S)$), whereas in the Moore FSM the output function is associated with each state: $\lambda\colon \mathbb{S} \rightarrow \mathbb{Y}$ ($Y = \lambda(S)$). Structure of the most popular FSM is presented in Fig. 1.

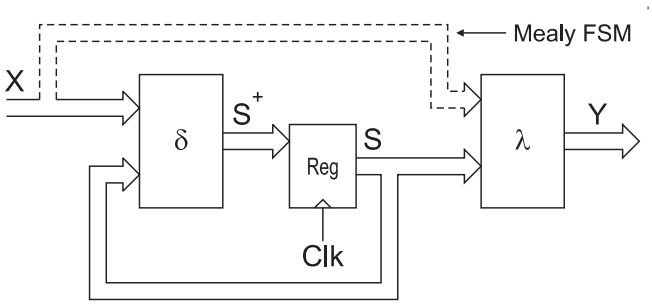

Fig. 1. The structure of an FSM

Generally the δ and λ functions are multi-output functions: $\delta = (\delta_{K-1}, \ldots, \delta_0)$, $\lambda = (\lambda_{m-1}, \ldots, \lambda_0)$, so let δ_i be i^{th} bit of the transition function and λ_j be j^{th} bit of the output function.

Internal states of an FSM are given mostly symbolic values. The goal of the state assignment is to assign to every state $S \in \mathbb{S}$ a binary representation $f\,(S)$, $f\colon \mathbb{S} \rightarrow \mathbb{B}^K$, where: K is the number of bits required to distinguish states. The minimum number of the code bits can be calculated from Eq. (1):

$$K = \lceil \log_2 l \rceil \tag{1}$$

where: $\lceil a \rceil$ is a minimum integer not less than a. The number of states must be greater than one. Otherwise the sequential circuit is just reduced to a combinatorial circuit (there is no need to assign states).

FSMs can be represented by a State Transition Table (STT). Every row of an STT corresponds to the transition between two states of the machine. The rows are divided into four columns corresponding to the primary inputs, present states, next states, and primary outputs (the *kiss* format). In the FSMs primary inputs and outputs are usually binary vectors, which may contain don't care entries. The present-state and the next-state columns are symbolic. The rows of a STT are called symbolic implicants (of the symbolic cover) [14]. A state transition graph, with corresponding STT, is presented on Fig. 2.

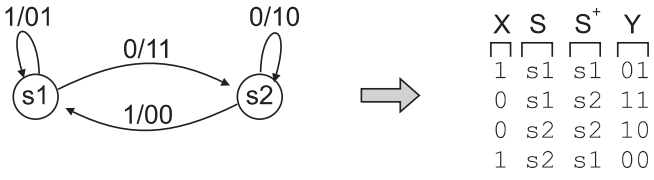

Fig. 2. State transition graph and corresponding STT

An assigned STT is a collection of multi-output implicants. An input part of a multi-output implicant corresponds to the primary input and a present state. An output part of a multi-output implicant corresponds to a next state and the primary output.

2.2. PAL-based CPLDs. A large majority of CPLDs are built of a simple cell matrix and a programmable interconnect array (PIA) – see Fig. 3.

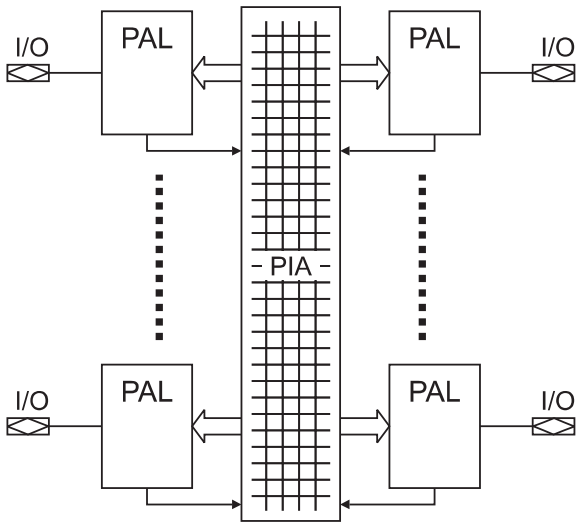

Fig. 3. A typical CPLD structure

The core of most CPLDs is a PAL-based cell. The generalized structure of the PAL-based cell is shown in Fig. 4. From the point of view of the presented method, three most important elements will be discussed.

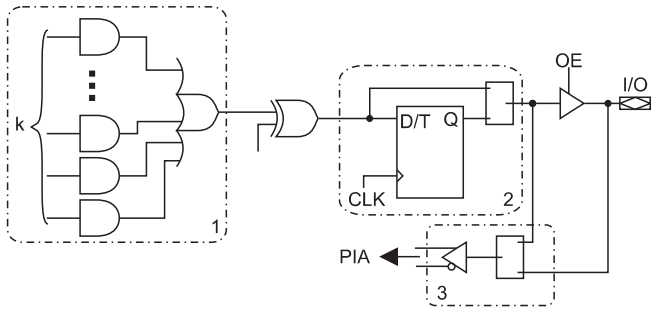

Fig. 4. A generalized structure of a PAL-based cell

A PAL-based cell contains a programmable-AND/fixed-OR structure (1), which can implement logic up to k product terms. In most cases $k < 8$ (usually 4 – Lattice: ispXPLD4A, or 5 – Xilinx: XC9500, MAX7000; Lattice: ispXPLD5000; Atmel: ATF1500). The output of an AND-gate connot be connected to more than one OR-gate.

The register (in some cases programmable as D or T flip-flop) can be bypassed for combinatorial operation (2). For a multi-level structure of the transition function, the last cell is of the synchronous type, while the previous cells are combinatorial. As a rule, the output function is combinatorial.

It was mentioned above, that sometimes functions are multi-level. It is required, when the number of implicants exceeds the number of AND-gates of the PAL-cell. Sometimes

it is possible to allocate additional term (or group of terms) from other cells to the OR-gate. Nevertheless, the implementation of every function is not possible. It is possible to expand the number of product terms thanks to the feedback lines (3). The D flip-flop's Q or $\overline{Q}$ output is fed back into the programmable AND-array, so sequential logic can be easily implemented. There are two basic types of product term expansion – Fig. 5.

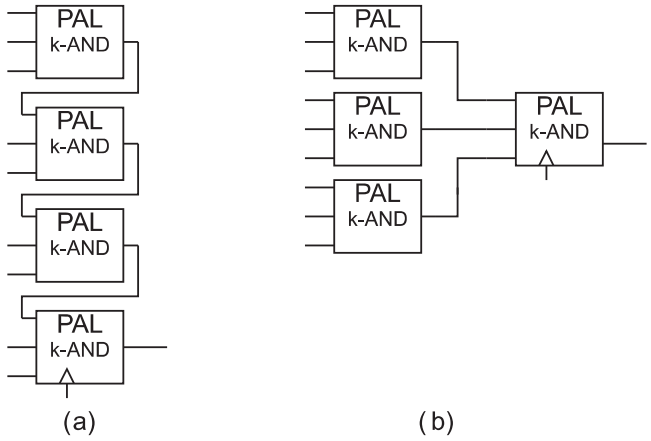

Fig. 5. Two types of product terms expansion

These two types of expansions use the same number of PAL-cells, but the second solution (Fig. 5b) is better with regard to the number of logic levels.

3. Definitions

3.1. State encoding using weights. Let the state weight η^{S_i} be a number of transits to the state S_i of the machine – the number of occurrences as a next state in STT.

Let η^{δ_i} be the number of implicants of a single transition function δ_i.

Let the μ-range be the number of bits equal to 1 in the code.

The distance $\nu(A, B)$ between two minterms A and B is the number of bits, they differ in. Let the $\nu(S_i, S_j)$ be a number of code bits assigned to states S_i and S_j, they differ in.

According to the definition, a coded STT is a collection of multi-output implicants: the input part of the multi-output implicant is the cube of those functions δ_i or λ_i, for which there is 1 on i^{th} position of the output part. To decrease the number of implicants:

1. Codes should be minimal with respect to μ-range.
2. States S_i with greater weights η^{S_i} should be assigned first.

The second conclusion is easy to explain – states that occur more frequently as a next state are assigned codes with a smaller number of logic "high". Going step forward, one more thing should be noticed: the state with the greatest weight should be assigned the code with all bits logic low ($\mu = 0$). This is because none of the single transition functions includes implicants corresponding to transition to the state.

Considering the FSM realization, dedicated for PAL-based CPLDs, the number of implicants of every single function should fit the number of product terms best. So, the number of implicants should be known in the process of state assignment. The total number of implicants of a single function δ_i or λ_i equals to the weight of states, for which there is a 1 on i^{th} position.

An example of a FSM is shown in Fig. 6. State $s2$ has the biggest weight – $\eta^{s2} = 6$. On the basis of the presented conclusions, the code 00 should be assigned to the state. Next, $s3$ ($\eta^{s3} = 3$) should be assigned the code 01 or 10, and then $s1$ or $s4$ ($\eta^{s1} = \eta^{s4} = 2$) with unused states. It is necessary to use 16 or 9 product terms to implement the transition function of the presented FSM.

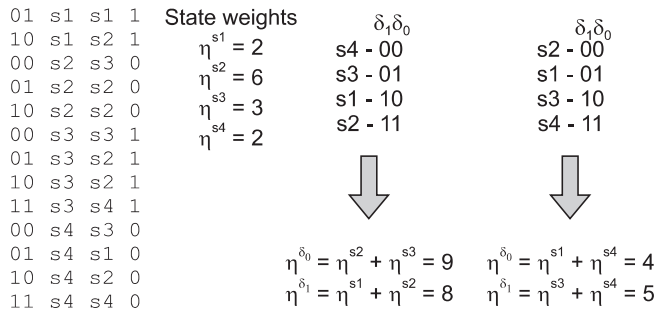

Fig. 6. Influence of state assignment on the number of implicants

Elements that refer to the weights of states have been proposed in [15].

Considerations presented in this subsection don't take into account two-level minimization. Of course the number of terms may be reduced. The main goal of the state assignment process should be to assign states with codes situated conveniently for the implicants to be merged. It is complicated for FSMs, because the input parts of the multi-output implicants are connected with the output part. The next state of the transition is the present state of another transition. Changing one bit of the state code involves changes in both input and output part of the implicants. On the other hand, elements of two-level minimization must be included in the state assignment process, in order to take advantage of the number of the PAL-based cell terms. Primary merging conditions and secondary merging conditions enable the algorithm to include elements of two-level minimization into the process of the state assignment. It is possible to predict the number of terms of a single function then.

3.2. Primary merging conditions. The idea of the state assignment is based on assigning to two states S_p and S_r, which correspond to the transitions to another state S_i for the same input X, binary codes that differ only in one position, $\nu(S_i, S_j) = 1$.

A fragment of an example FSM with two different state assignment is shown in Fig. 7. There are two transitions presented in figure. The state $s3$ is the next state for both transitions. The inputs and the outputs are also the same for both transitions. The present states are $s1$ in first transition and $s2$ in the second transition. The state $s2$ should be assigned the code, such as the distance to the state $s1$ code is one ($\nu(s1, s2) = 1$).

Two presented multi-output implicants can be merged into one implicant (right branch in figure). The distance for the case on the left branch in figure is $\nu(s1, s2) = 2$. Implicants cannot be merged.

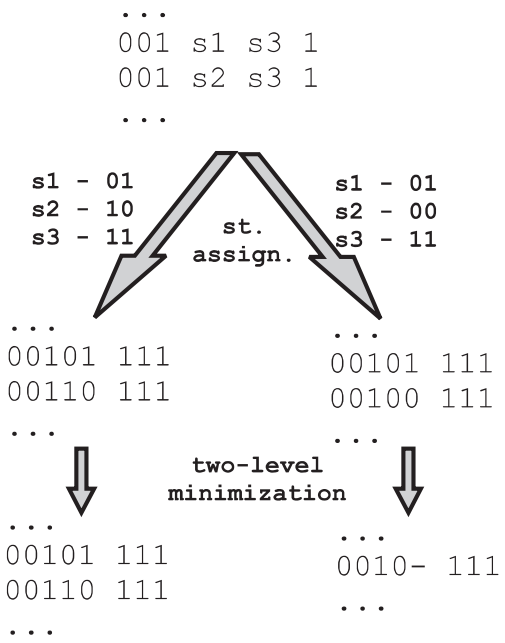

Fig. 7. Fragment of an example FSM with two types of state assignment

DEFINITION 1. A Primary Merging Condition (PMC) $\{S_p, S_r\}_X^{S_i}$ for a transition function is a condition formed by two transitions from states S_p and S_r to the state S_i that correspond to the same input X.

DEFINITION 2. A Primary merging condition $\{S_p, S_r\}_X^{\lambda_i}$ for the output function is a condition formed by two transitions from states S_p and S_r, for which the output function λ_i is 1, that correspond to the same input X.

To satisfy primary merging conditions, states S_p and S_r have to be assigned binary codes, whose distance equals one.

Primary merging conditions $\{s1, s2\}_{001}^{s3}$ and $\{s1, s2\}_{001}^{\lambda 0}$, presented in Fig. 8, concern to fragment of an FSM presented in Fig. 7.

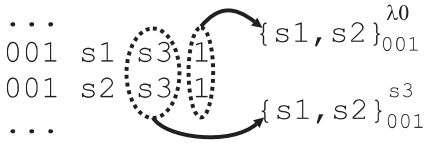

Fig. 8. A part of STT with primary merging conditions

3.3. Secondary merging conditions. Product terms of the PAL-based cell cannot be shared among the functions. So the structure extorts independent realization of every function $f_i \colon \mathbb{B}^n \to \mathbb{B}$ for $i = m-1 \ldots 0$. The two-level minimization is carried out for every function f_i independently (each function is minimized one at a time as a single-output function).

As a result of the state assignment, the transition function δ_i can contain implicants, the distance of which is 1, but not as the effect of satisfying primary merging conditions. This can happen, if the transition function contains implicants that refer to:

– transitions from two different actual states S_i and S_j, that are carried out for the same input x_u, if the distance between the codes of those states equals one – $\nu(S_i, S_j) = 1$,
– transitions from the same state S_i for two different inputs X_u and X_w, the distance between which is also one – $\nu(X_u, X_w) = 1$.

Consider the example shown in Fig. 9. No primary merging conditions exist for the presented fragment of the unassigned STT. The states are assigned codes and then the list of multi-output implicants is splitted to single-output implicants (because product terms of a PAL-based cell cannot be shared among the functions). The list of implicants is reduced to two after the two-level minimization. One pair of implicants is merged because there is pair of transitions from the states $s1$ and $s3$ for the same input 01 and the output has a 1 on the same position $\delta 2$. It is of course possible because the distance between codes of the states $s1$ and $s3$ equals one.

The second pair of implicants can be merged because there are transitions from the state $s3$ for two different inputs 01 and 11, the distance between which is one ($\nu(X_u, X_w) = 1$) and two implicants that correspond to the transitions belong to the same function $\delta 1$.

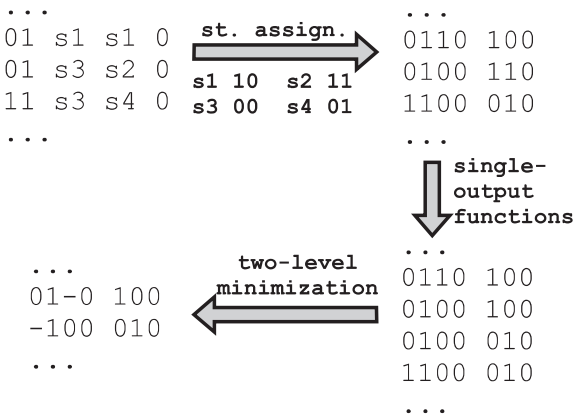

Fig. 9. A part of an STT before and after the state assignment process

DEFINITION 3. A Secondary Merging Condition (SMC) $\{S_p, S_r\}_{\delta_i, X}^{S_a, S_b}$ is a condition that is formed by two present states S_p and S_r from which there are transitions to next states S_a and S_b for the same input X. The symbolic implicants, referring to the present states S_p and S_r, belong to the same transition function δ_i.

To satisfy the secondary merging conditions $\{S_p, S_r\}_{\delta_i, X}^{S_a, S_b}$, the states S_p and S_r have to be assigned binary codes with the distance between them equal to one – $\nu(S_p, S_r) = 1$.

DEFINITION 4. A Secondary Merging Condition $\{S_p\}^{S_a,S_b}_{\delta_i,X_u,X_w}$ is a condition that is formed by the present state S_p, from which there are transitions to the next states S_a and S_b for inputs X_u and X_w. The symbolic implicants, referring to the present state S_p, belong to the same transition function δ_i.

The Secondary merging condition $\{S_p\}^{S_a,S_b}_{\delta_i,X_u,X_w}$ is always fulfilled, and two implicants are merged. The condition is written in order to eliminate multiple merging of the same implicants.

SMCs emerge during the process of state assignment. One step of thestate encoding process is shown in Fig. 10 . The mechanism of the SMC arising is also presented.

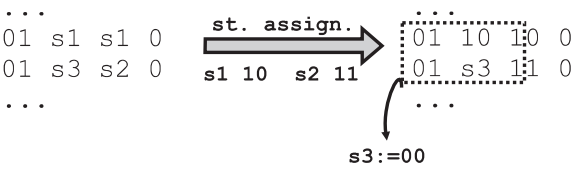

Fig. 10. The mechanism of the SMCs forming, during the process of state assignment

3.4. The implicants distribution table. The basic difficulty of an effective term using, when functions are to be implemented in PAL-based devices, is two-level minimization. As a rule it is carried out after the state assignment process, so the result cannot be foreseen. The elements of the two-level minimization or methods of counting the number of implicant (as the effect of the minimization process) have to be included in the process of state assignment. It is easy to write primary merging conditions, but secondary merging conditions appear only in the state assignment process, and come from the distribution of implicants among single functions.

DEFINITION 5. The Implicants Distribution Table (IDT) T is the table divided into columns, corresponding to the weights η^{δ_i} of the single functions δ_i. Every row of the table corresponds to the number of implicants which is equal to weights of the states. The weights of the states are written into those columns η^{δ_i}, for which there is a 1 on i^{th} position of the code.

When the PMC or SMC is fulfilled, a -1 is written into column corresponding to the function δ_i, for which two implicants are merged.

Example of an IDT is shown in Fig. 11.

4. The method

The way of expanding the product term number is to feed back an OR-gate output to the logic array. Two basic types of expansion are presented on Fig. 5. The main cost of such an implementation is reduction of the system speed caused by adding extra logic levels to the structure. The delay of forming an output vector in an FSMs depends not only on the speed of an output block. It depends on the speed of a block which realizes the transition function, too (Fig. 1.). The speed of the transition block is much more critical, because the state vector is formed synchronously, while the output vector is formed asynchronously (the output block is combinatorial). The non-uniform increase of logic levels of the single transition function should be avoided – the logic level number of the transition block ξ_δ equals to the number of cells used in the longest path.

The number of logic levels number of fast automata must be as small as possible. The logic level extraction problem is solved in the presented approach.

It is possible to determine the number of logic cells of the transition block after the state assignment process. Let the σ^{δ_i} be the number of logic cells of the transition block:

$$\sigma^{\delta_i} = \begin{cases} 1 & \text{dla } \eta^{\delta_i} = 1 \\ \left\lceil \frac{\eta^{\delta_i}-1}{k-1} \right\rceil & \text{dla } \eta^{\delta_i} > 1 \end{cases} \tag{2}$$

where: k is the number of the product terms in the PAL-cell ($k > 1$).

Let the ξ_δ be the number of logic levels of the transition block. The number of logic levels for the expansion strategy presented in Fig. 5a, equals to the number of logic cells calculated on the basis of Eq. (2) in the longest path: $\xi_\delta = \max_i(\eta^{\delta_i})$.

The number of logic levels for the expansion strategy presented in Fig. 5b can be calculated from Eq. (3).

$$\xi_\delta = \begin{cases} 1 & \text{if } \eta^{\delta_i} < k \\ \max_i(\lceil \lg_k \eta^{\delta_i} \rceil) & \text{if } \eta^{\delta_i} \geq k \end{cases} \tag{3}$$

The question is: is it possible to estimate the minimum number of logic levels of the transition block, for which the realization is possible? The answer is yes. It is so important because the number of logic levels of the transition block must be known in advance – before the state assignment process. It can be determined from the equation (4).

$$\xi_\delta = \begin{cases} 1 & \text{if } \eta^{S_i} < k \\ \lceil \lg_k \eta^{S_i} \rceil & \text{if } \eta^{S_i} \geq k \end{cases} \tag{4}$$

where: η^{S_i} is the greatest but one weight (unless there are two, or more states with the same greatest weight). The state with the greatest weight is assigned the zero code, so none of the functions has implicants corresponding to transitions to the state.

The main idea is to count the number of logic levels of every single transition block during the state assignment process. In following steps of the algorithm, unassigned state with the greatest weigh, is assigned a minimum μ-range code. If the number of logic levels exceeds the assumption, the number of codding bits is increased. Codes already assigned to states are supplemented with 0.

The algorithm (ml):

1. Calculate the number K of bits of coding word (equation (1)).
2. Specify the PMCs of the transition function.
3. Assign to the state with the greatest weight η^{s_i} the zero code ($\mu = 0$). If there is more than one state that satisfies the condition, choose the state s_i which can satisfy most PMCs $\{s_i, s_r\}^{s_j}_x$.

4. $\mu := 1$.
5. Calculate the number ξ_δ of logic levels of the transition function (Eq. (4)).
6. Choose the state with the greatest weight η^{s_i}. If there is more than one state that satisfies the condition, the sort key is as follows:
   (a) choose the state s_i, which can satisfy more primary merging conditions $\{s_i, s_r\}_x^{s_j}$,
   (b) choose the state s_i, which can satisfy more non-excluding secondary merging conditions $\{s_i, s_r\}_{\delta_j,x}^{s_a,s_b}$,
7. If none of the μ-range codes is free, $\mu := \mu + 1$.
8. Assign to the chosen state s_i a free code of the μ-order; if there is more than one possibility, the sort key is as follows:
   (a) the number of PAL-based cell incrementation is the smallest,
   (b) the sum of all η^{δ_i} is the smallest,

   (The PAL-cell incrementation and the sum of all η^{δ_i} are calculated after making allowance for every satisfied merging condition)
9. If exists $\delta_i : \xi_i > \xi_\delta$, than:
   (a) cancel the last assignment,
   (b) $K := K + 1$,
   (c) $\mu := 1$,
   (d) suplement the already assigned codes with 0 on the MSB,
   (e) return to point 8.
10. Refresh the IDT.
11. Revise the secondary merging conditions.
12. Cancel the satisfied or the excluded primary and secondary merging conditions.
13. If not all states have been already encoded, than return to point 6.
14. Choose the output level activity [16].
15. End.

EXAMPLE. Let's consider an example. The STT of the example FSM is given in Fig. 11 (*kiss2* format don't care states are denoted by '*'; A current state don't care condition indicates that no matter what state you are in, a specified input produces a transition to a given next state and output condition). On the basis of the presented STT, the weights and the PMCs are specified. The coding length K is 4. It has been assumed that the product term number of the PAL-based cell is 3. The number of logic levels is determined on the basis of weight of the state $s1$ (or $s3$) and equals one.

The state $s0$ is assigned first of all – the weight η^{s0} is the greatest. According the algorithm (and drawn conclusions) the state $s0$ is assigned 0000. Next, states are assigned respectively $s1 - 0001$, $s3 - 0010$, $s4 - 0100$ and $s2 - 1000$. According to the definition 5, the weights of states are written into those columns δ_i, for which there is a 1 on the i^{th} position of the code. Four rows of the presented part of the IDT correspond to the numbers of implicants, which are equal to weights of the states. The situation is shown in Fig. 12a.

```
0-- *  s0 00
111 s1 s2 10
110 s1 s1 10
111 s2 s3 00
100 s3 s4 11
101 s3 s5 11
111 s3 s3 11
100 s4 s1 01
101 s4 s3 01
110 s4 s4 01
100 s5 s1 10
111 s5 s6 10
111 s6 s7 11
111 s7 s8 00
111 s8 s9 11
101 s9 s2 01
```

Weights

$\eta^{s0} = 10$
$\eta^{s1} = 3$
$\eta^{s2} = 2$
$\eta^{s3} = 3$
$\eta^{s4} = 2$
$\eta^{s5} = \eta^{s6} = \eta^{s7} = \eta^{s8} = \eta^{s9} = 1$

PMC:
$\{s4, s5\}_{100}^{s1}$
$\{s2, s3\}_{111}^{s3}$

Fig. 11. An example function with weights and PMCs

$\eta^{\delta3}$	$\eta^{\delta2}$	$\eta^{\delta1}$	$\eta^{\delta0}$	st.
0	0	0	0	s0
			3	s1
		3		s3
	2			s4
2				s2
2	2	3	3	**sum**

(a)

$\eta^{\delta3}$	$\eta^{\delta2}$	$\eta^{\delta1}$	$\eta^{\delta0}$	st.	
0	0	0	0	s0	
			3	s1	
		3		s3	
	2			s4	
2				s2	
1	1			s5	
			-1	$\{s4, s5\}_{100}^{s1}$	
		-1		$\{s3\}_{\delta2,100,101}^{s4,s5}$	
3	2	3	2	**sum**	$\{S3, s9\}_{\delta3,101}^{s2,s5}$

(b)

$\eta^{\delta3}$	$\eta^{\delta2}$	$\eta^{\delta1}$	$\eta^{\delta0}$	st.	
0	0	0	0	s0	
			3	s1	
		3		s3	
	2			s4	
2				s2	
	1	1		s5	
			-1	$\{s4, s5\}_{100}^{s1}$	
		-1		$\{s3\}_{\delta2,100,101}^{s4,s5}$	
	-1			$\{s3\}_{\delta1,101,111}^{s3,s5}$	
2	2	3	2	**sum**	$\{s3, s4\}_{\delta1,101}^{s3,s5}$

(c)

Fig. 12. State assignment process

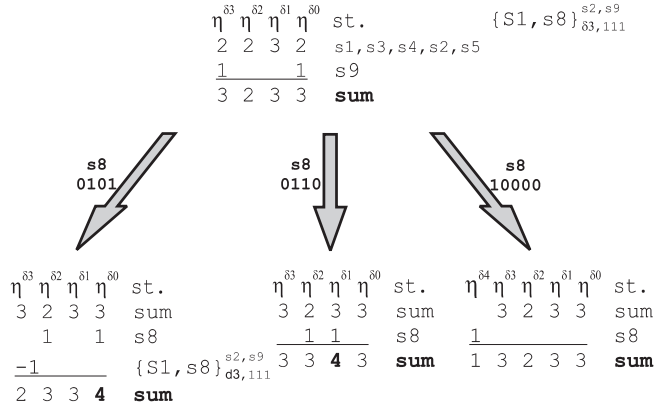

Fig. 13. An example function with IDTs being the effects of the different state assignment

The continuation of the state assignment procedure is presented in Fig. 12b and c. First, the state $s5$ is assigned 1100. The PMC $\{s4, s5\}_1^{s1}\ 00$ is fulfilled, so a -1 is written into ITD for the column corresponding to the function δ_0 (η^{δ_0} is decremented). It should be noticed that states $s4$ and $s5$ have a common logic high on the position corresponding to the function δ_2. Because there are two transitions from the state $s3$ for inputs 100 and 101 ($\{s3\}^{s4,s5}_{\delta_2,100,101}$ is fulfilled), η^{δ_0} is also decremented.

The state $s5$ can be assigned 0110 as well (Fig. 12c). The two conditions are satisfied like in Fig. 12b. But there is one more SMC for this case. States $s3$ and $s5$ have a common 1 on the position corresponding to the function δ_1. There are two transitions from the state $s3$ for inputs 101 and 111 ($\{s3\}^{s3,s5}_{\delta_1,101,111}$), so the η^{δ_1} is decremented too.

A starting point to assign state $s8$ is an IDT form Fig. 12c. Next, the state $s8$ should be assigned the code for which the distance between the state $s8$ and the state $s1$ is one. Assigning to the state the code 0101 makes the structure of the FSM two-level. If whichever of the free codes was chosen, there would be the same effect (like in the one shown in Fig. 13, after the state $s8$ was encoded with 0110). The main idea in this situation is to use an additional bit and to assign the state $s8$ code 10000. The number of logic cells is the same as in previous cases, but the number of the logic levels still remains one. Codes, that are already used, are supplemented with 0 on the MSB position.

5. Experimental results

The experiments were carried out by means of:

- JEDI [12]: the input dominant algorithm (**i**), the output dominant algorithm (**o**) and the coupled dominant algorithm (**c**);
- NOVA [6]: the input and output (dominance) constraints (iohybrid_code – **ioh**), the input constraints (ihybrid_code – **ih**) and the input constraints (iexact_code – **ie**);
- the "one-hot"encoding (**one**);
- the presented ml-algorithm (**ml**).

Table 1
Comparison of the ml-algorithm with JEDI and NOVA

B-mark	$k=3$							$k=4$							$k=5$						
	i	o	c	ioh	ih	ie	ml	i	o	c	ioh	ih	ie	ml	i	o	c	ioh	ih	ie	ml
The number of logic levels of the transition function																					
bbtas	2	2	2	2	2	2	1	2	2	2	2	2	2	1	2	1	1	2	1	1	1
dk27	1	2	1	2	2	2	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1
ex3	2	2	2	2	3	-	2	2	2	2	2	2	-	1	2	2	2	2	2	-	1
ex5	2	2	2	3	2	2	2	2	2	2	2	2	2	1	2	2	1	2	2	2	1
ex7	2	2	2	3	2	2	2	2	2	2	2	2	2	1	2	2	2	3	2	2	1
lion	1	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
lion9	1	2	1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
mc	2	1	1	2	2	2	1	1	1	1	2	1	2	1	1	1	1	1	1	1	1
train11	2	2	2	2	2	-	1	2	2	2	2	2	-	1	1	2	1	1	1	-	1
train4	1	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Total	16	18	17	21	20	15[1]	13	15	15	15	17	16	13[1]	10	14	14	12	14	14	10[1]	10
The number of logic cells of the transition function																					
bbtas	6	2	3	5	5	5	5	5	2	2	4	4	4	5	4	2	2	4	4	4	4
dk27	3	4	3	6	5	5	3	3	3	3	5	5	5	3	3	3	3	3	3	3	3
ex3	12	3	3	12	2	2	8	9	2	2	9	2	2	6	7	2	2	7	2	2	4
ex5	12	6	4	17	3	4	8	8	4	4	12	2	2	5	8	4	4	9	2	2	4
ex7	14	5	6	13	6	-	9	10	5	5	9	5	-	5	8	3	3	8	4	-	5
lion	2	2	2	3	11	12	3	2	2	2	2	8	8	2	2	2	2	2	6	8	2
lion9	4	7	6	4	3	4	8	4	5	6	4	2	4	7	4	5	4	4	2	2	5
mc	3	9	14	3	13	13	2	2	6	10	3	9	9	2	2	5	8	2	8	7	2
train11	7	9	8	7	6	6	6	5	7	7	5	4	4	6	4	6	4	4	3	3	4
train4	2	11	10	3	14	-	2	2	7	8	2	9	-	2	2	7	6	2	8	-	2
Total	65	58	59	73	68	51[1]	54	50	43	49	55	50	38[1]	43	44	39	38	45	42	31[1]	35

[1] – without ex3 and train11 benchmarks

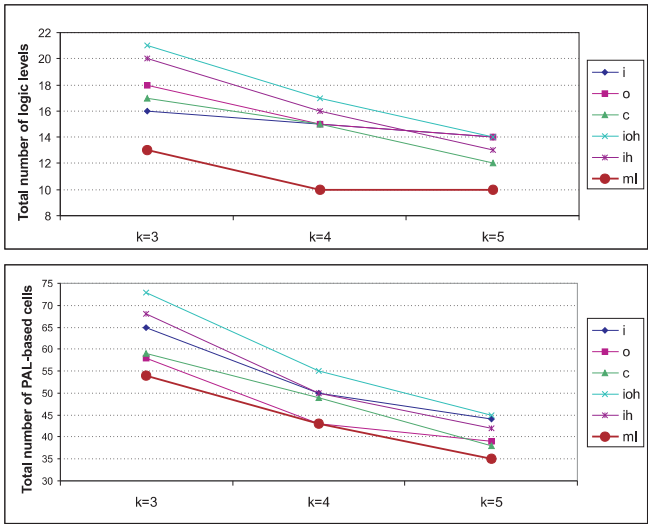

Fig. 14. Comparison of different algorithms targeting at reduction of the number of logic levels

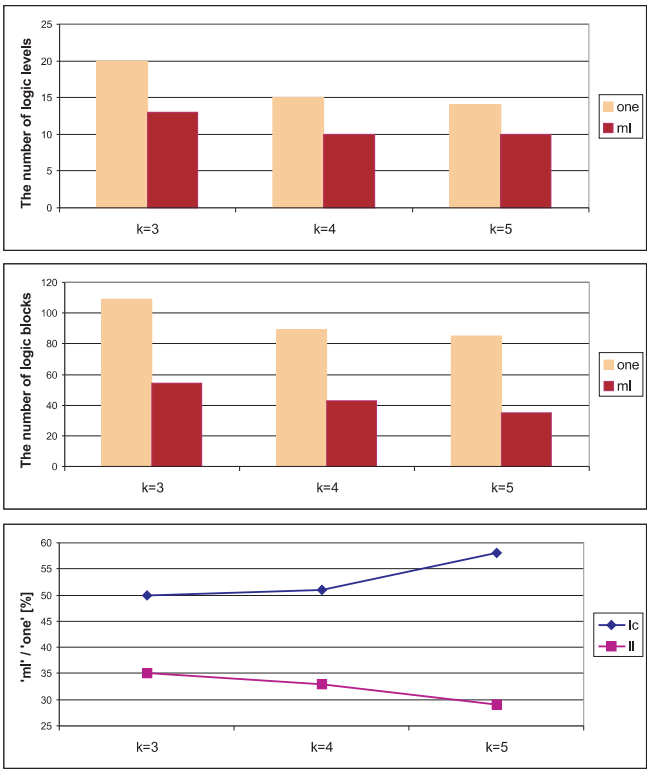

Fig. 15. Comparison of the "one-hot" coding with the proposed method: lc – the number of logic cells, ll – the number of logic levels

Experiment were carried out using some selected benchmarks [17].

Experimental results for NOVA, JEDI, and the considered state assignment targeting at fast automata (the ml-algorithm) are shown in Table 1. The ml-algorithm gave the best results for all analyzed logic cells. The total number of logic levels is reduced by about 18%, 33%, and 16% for 3-, 4- and 5-terms cells (for the worst case). It can be observed, that for $k = 4$ and $k = 5$, for all presented benchmarks transition blocks are one-level. Moreover, the minimization of the number of logic levels is not relevant with the number of cell expansion. The considerable reduction of the number of logic levels guaranteed a comparable number of logic cells used to implement the transition function.

The graphs presented on Fig. 14 illustrate the comparisons of the total numbers of logic levels and logic cells obtained for different algorithms.

Table 2
Comparison of the ml-algorithm with "one-hot" coding

B-mark	$k=3$		$k=4$		$k=5$	
	one	ml	one	ml	one	ml
bbtas	2/11	1/5	2/7	1/5	2/7	1/4
dk27	2/8	1/3	1/7	1/3	1/7	1/3
ex3	3/19	2/8	2/14	1/6	2/13	1/4
ex5	3/16	2/8	2/13	1/5	2/12	1/4
ex7	3/19	2/9	2/15	1/5	2/14	1/5
lion	1/4	1/3	1/4	1/2	1/4	1/2
lion9	1/9	1/8	1/9	1/7	1/9	1/5
mc	1/4	1/2	1/4	1/2	1/4	1/2
train11	2/12	1/6	2/12	1/6	1/11	1/4
train4	2/7	1/2	1/4	1/2	1/4	1/2
Total	20/109	13/54	15/89	10/43	14/85	10/35

a/b: a – the number of logic levels
b – the number of logic cells

Table 3
Comparison of different algorithms for the expansion strategy like in Fig. 15a

B-mark	$k=3$			$k=4$			$k=5$		
	one	ih	ml	one	ih	ml	one	ih	ml
bbtas	3	2	1	2	2	1	2	1	1
dk27	2	2	1	1	2	1	1	1	1
ex3	8	5	2	5	3	1	4	3	1
ex5	7	4	2	5	3	1	4	2	1
ex7	9	4	2	6	3	1	5	2	1
lion	1	1	1	1	1	1	1	1	1
lion9	1	2	1	1	1	1	1	1	1
mc	1	2	1	1	1	1	1	1	1
train11	2	2	1	2	2	1	1	1	1
train4	2	2	1	1	1	1	1	1	1
Total	36	26	13	25	19	10	21	14	10

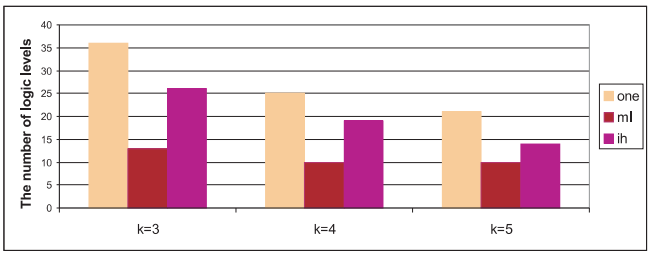

Fig. 16. Comparison of different algorithms for the expansion strategy like in Fig. 15a

In Table 2 the comparison of the ml-algorithm with "one-hot" encoding for the chosen benchmarks is shown. The numbers of logic levels are calculated from Eq. (3) and concern the

transition block (the expansion strategy like in Fig. 5b). The proposed algorithm gives much better results than the "one-hot" method, which is considered as a good method for fast automata. Moreover, increase of the number of product terms in PAL-based cell leads to decreasing the ratio of the number of logic levels between the ml-algorithm and the "one-hot" algorithm, while the ratio of the number of logic cells increases. This can be explained by better fitting the structure of an automaton after the state assignment using the ml-algorithm than using the "one-hot" algorithm.

The comparison of total numbers of PAL-cells based on the Table 2 is shown on a graph in Fig. 15.

The comparison of the ml-algorithm with the "one-hot" and JEDI ih-algorithm is presented in Table 3 and in Fig. 16. The numbers of logic levels of the transition blocks are calculated from Eq. (2) – the expansion strategy like in Fig. 15a.

The main disadvantage of the "one-hot" method is that there is no zero code. Every of the δ_i function includes implicants that correspond to transitions to the state S_i. In the group of benchmarks, there are three with disproportional weight of one state (*ex3, ex5, ex7*). Assigning to those states zero codes provide solutions that are faster and use less terms. The "one-hot" method may be than improved by expanding the set of codes with the zero code.

6. Conclusions

The method proposed in the paper method matches the structure of sequential automata to the PAL-based CPLDs better. The novel method based on Primary and Secondary Merging Conditions and Implicants Distribution Table makes that the limitations of PAL-based cell are taken into account at an early stage of synthesis.

The one-hot coding is considered to be a method, which generates fast automata. The preliminary experimental results don't confirm this thesis. It is known that generally the number of code bits (in "one-hot" coding) is redundant – it may require too many bits to be practical. The modification of the "one-hot" method, presented in this paper, may lead to faster structures.

Experimental results indicate that the ml-algorithm is very efficient. The algorithm is also very fast – faster than NOVA and JEDI (e.g. for ex3: $t_{ioh}/t_{mb} \gg 10$).

Acknowledgements. This work has been supported by the Ministry of Scientific Research and Information Technology of the Republic of Poland (Grant No. 3T11B04027).

REFERENCES

[1] G. Saucier, P. Sicard, and L. Bouchet, "Multi-level synthesis on PAL's", *Proc. European Design Automation Conference*, Glasgow, 542–546 (1990).

[2] G. Saucier, P. Sicard, and L. Bouchet, "Multi-level synthesis on programmable devices in the ASYL system", *Euro ASIC*, 136–141 (1990).

[3] D. Kania, "A technology mapping algorithm for PAL-based devices using multi-output function graphs", *Proc. 26-th Euromicro Conference*, 146–153, Maastricht (2000).

[4] D. Kania, "Logic synthesis of multi-output functions for PAL-based CPLDs", *IEEE Int. Conf. Field-Programmable Technology*, 429–432, Hong Kong (2002).

[5] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines", *IEEE Trans. on CAD/ICAS* CAD-4 (3), 269–284 (1985).

[6] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment for finite state machines for optimal two-level logic implementation", *IEEE Trans. on Computer-Aided Design* 9, 905–924 (1990).

[7] E. Sentovich, K. Singh, L.Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis", *Proc. Int. Conf. on Computer Design*, 328–333 (1992).

[8] S. Yang and M. Ciesielski. "Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization", *IEEE Trans. on Computer-Aided Design* 10, 4–12 (1991).

[9] S. Devadas, A.R. Newton, and P. Ashar. "Exact algorithms for output encoding, state assignment and four-level boolean minimization", *IEEE Trans. on Computer-Aided Design* 10, 13–27 (1991).

[10] M. Chyzy and W. Kosinski, "Evolutionary algorithm for state assignment of finite state machines", *Proc. of Euromicro Symposium on Digital System Design*, 359–362 (2002).

[11] S. Devadas, H.K. Ma, R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations", *IEEE Trans. on Computer-Aided Design* 7 (12), 1290–1300 (1988).

[12] B. Lin and R. Newton, "Synthesis of multiple level logic from symbolic high-level description languages", *Proc. Int. Conf. on VLSI*, 187–206 (1989).

[13] P. Sicard, M. Crastes, K. Sakouti, and G. Saucier, "Automatic synthesis of boolean functions on xilinx and actel programmable devices", *Proc. Euro ASIC'91*, Paris, 1991.

[14] T. Villa, T. Saldanha, A. Brayton, and A. Sangiovanni-Vincentelli, "Symbolic two-level minimization", *IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems* 16 (7), 692–708 (1997).

[15] P.K. Lala. "An algorithm for the state assignment of synchronous sequential circuits", *Electronics Letters* 14 (6), 199–201 (1978).

[16] R. Czerwinski and D. Kania, "FSM's state assignment method based on level of output activity", *RUC'2003*, 9–16, Szczecin (2003).

[17] MCNC, "LGSynth'91 benchmarks", Collaborative Benchmarking Laboratory, Department of Computer Science, North Carolina State University, `http://www.cbl.ncsu.edu/`.