Int. J. Appl. Math. Comput. Sci., 2009, Vol. 19, No. 4, 647–659
DOI: 10.2478/v10006-009-0052-0

SYNTHESIS OF FINITE STATE MACHINES FOR CPLDs

ROBERT CZERWIŃSKI, DARIUSZ KANIA

Institute of Electronics
Silesian University of Technology, ul. Akademicka 16, 44–100 Gliwice, Poland
e-mail: {robert.czerwinski,dariusz.kania}@polsl.pl

The paper presents a new two-step approach to FSM synthesis for PAL-based CPLDs that strives to find an optimum fit of an FSM to the structure of the CPLD. The first step, the original state assignment method, includes techniques of two-level minimization and aims at area minimization. The second step, PAL-oriented multi-level optimization, is a search for implicants that can be shared by several functions. It is based on the graph of outputs. Results of experiments prove that the presented approach is especially effective for PAL-based CPLD structures containing a low number of product terms.

Keywords: logic synthesis, FSM, state assignment, logic optimization, CPLD.

1. Introduction

Most CPLDs (Complex Programmable Logic Devices) have architectures consisting of a PAL-like AND-OR structure and multi-level logic capability combined with an integrated array of logic and I/O macrocells. Whereas each CPLD has an original internal structure, its kernel is in most cases formed by a PAL-based structure (Sharma, 1998). PAL-based logic blocks usually have a strictly defined number of terms connected to the individual output macrocells. This feature of a PAL based block significantly affects the synthesis process of digital circuits based on such devices. The essence of CPLD-oriented synthesis of FSMs (Finite State Machines) using PAL-based devices is the implementation of the logic function using logic blocks including k terms (Fig. 1).

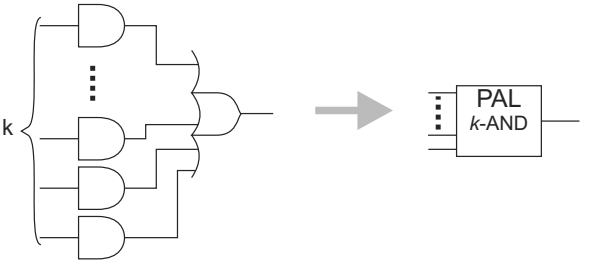

Fig. 1. Structure and symbol of a PAL-based logic cell.

One of the most important stages of the FSM design flow is state assignment. There are many methods of state assignment. Some are considered optimal (Sentovich *et al*., 1992; Villa and Sangiovanni-Vincentelli, 1990). The state assignment problem is often solved together with input and output encoding (Yang and Ciesielski, 1991). Some methods are based on dichotomies (Villa and Sangiovanni-Vincentelli, 1990) or dominance graphs (Devadas and Newton, 1991). Sometimes the problem is solved using genetic algorithms (Chyży and Kłosiński, 2002). There are also many aims of finite state machine synthesis, like reducing power consumption of automata (Mengibar *et al*., 2005; Chattopadhyay, 2001) or synthesis for testability (Park *et al*., 2000). Sometimes, the main idea of logic synthesis is based on functional (Jóźwiak and Volf, 1995) or structural decomposition of FSMs (Barkalov *et al*., 2007). These methods are composed for different structures of Mealy and Moore automata (Salauyou *et al*., 2006). A vast majority of methods are dedicated for automata which are to be implemented in PLA-based devices (Baranov, 1994; Villa *et al*., 1997). Some well-known methods like "one-hot" coding or binary coding, are still part of vendor tools, even though they give results far from optimum.

The aim of the proposed state assignment method is to minimize the number of PAL-based macrocells by fitting the FSM to the structure of the CPLD in the best possible way (Czerwiński and Kania, 2005; Czerwiński, 2006; Czerwiński *et al*., 2006). The elements of two-level minimization are included in the state assignment process. Primary and secondary merging conditions, and the implicants distribution table are introduced

in the paper. In the final stage, the procedure of code exchange is carried out in order to aid multi-level optimization (Kania, 2003).

This paper is structured as follows: Section 2 introduces some basic information about state assignment and multi-level optimization. Section 3 focuses on the synthesis of FSMs for PAL-based CPLDs. Experimental results are reported in Section 4. The paper closes with conclusions in Section 5.

2. Basic definitions

The mathematical model of a sequential circuit is a finite state machine, which is represented by a quintuple, $\{X, Y, S, \delta, \lambda\}$, where X is a finite input alphabet, Y is a finite output alphabet, S is a finite set of states, δ is the transition function, λ is the output function. The transition function of an FSM determines the next state of the automata (S^+).

Let the state weight η^{s_i} be the number of transits to the state s_i of the machine—the number of occurrences as a next state in State Transition Table (STT). Let the μ-range be the number of bits 1 in the code. The distance $\nu(A, B)$ between two minterms A and B is the number of bits they differ in. Let $\nu(S_i, S_j)$ be the distance between codes of the states S_i and S_j.

Let f be a multi-output logic function $f\colon B^n \to B^m$, where $B = \{0, 1\}$. The classical method of implementation of the function $f\colon B^n \to B^m$ within PAL-based structures is related to the implementation of the minimised functions $f\colon B^n \to B^1$ $(i = 1, 2, \ldots, m)$ by means of PAL-based logic blocks consisting of k-terms. Let the discriminant Δ_{f_i} be the number of those implicants, provided the function $f\colon B^n \to B^1$ constitutes true values (e.g., Δ_{δ_i} is the number of implicants of the δ_i function).

Let σ_{f_i} denote the number of logic blocks necessary for the implementation of the i-th function. In the case when $f_i > k$, the implementation of the f_i function by means of PAL-based logic blocks consisting of k-terms needs the realization of feedback loops. Therefore, the number of $\sigma_{f_i} = \lceil (\Delta_{f_i} - k)/(k-1) \rceil + 1$ PAL-based logic blocks consisting of k-terms will be used, where $\lceil x \rceil$ denotes the lowest integer number, not less than x. For classical implementation of m-functions (every function has been minimized separately), the implementation of σ_f^1 PAL-based logic blocks is necessary, where

$$\sigma_f^1 = \sum_{i=1}^{m} \left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right). \tag{1}$$

Example 1. Let $f_1 = b\bar{c}\bar{d} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + \bar{b}cd$ and $f_0 = \bar{a}\bar{b}d + ab\bar{d} + bc\bar{d} + \bar{b}cd$. Separate implementation of the f_1 and f_2 function requires four logic PAL-based blocks, which include three terms. However, another implementation is also possible. The implementation presented in Fig. 2 requires only three blocks. ♦

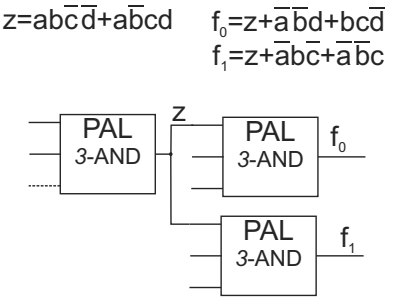

Fig. 2. Implementation of the example two-output function.

The proposed PAL-based multi-level optimization allows a considerable reduction of the number of required logic blocks. The main idea of the method is based on searching for shared mutli-output implicants.

3. Synthesis of FSMs for PAL-based devices

The proposed logic synthesis process consists of two main procedures:

- PAL-oriented state assignment,
- PAL-oriented multi-level optimization.

An overview of the logic synthesis system is shown in Fig. 3.

3.1. PAL-oriented state assignment. A coded STT is a collection of multi-output implicants. The total number of implicants of the single function δ_i or λ_i equals the weight of states for which there is a 1 on i-th position. This brings to mind that codes with a minimal μ-range

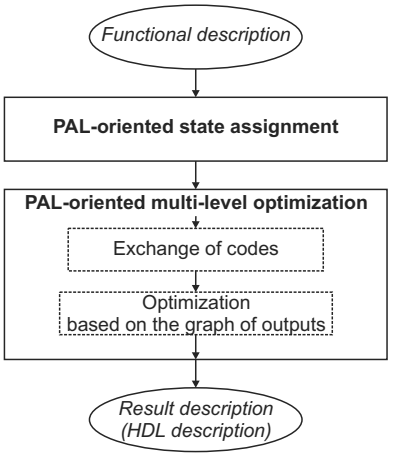

Fig. 3. Design flow of the proposed logic synthesis of FSMs for PAL-based devices.

should be used to encode states. Moreover, states s_j with greater weights η^{s_j} should be coded first of all, because those states occur more frequently as next. In that way, codes with the smallest number of a logic high occur more frequently. Of course, the state with the greatest weight is assigned the code with all bits logic low ($\mu = 0$), so none of the single transition functions includes implicants corresponding to the state.

The number of terms may be reduced after two-level minimization. The main goal of state assignment should be to assign states to codes situated suitably to each other in the Boolean space, so some implicants could be merged after two-level minimization. This gets complicated in FSMs because the input parts of multi-output implicants are connected with the output part. The next state of one transition is the present state of another. Changing one bit of the state code involves changes in both the 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 PAL-cell terms. Primary and secondary merging conditions enable to include elements of two-level minimization in the process of state assignment (Czerwiński and Kania, 2005; Czerwiński, 2006; Czerwiński *et al*., 2006).

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

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

The idea of state assignment is based on assigning to two states S_p and S_r of PMC binary codes that differ only in one position, $\nu(S_p, S_r) = 1$. A fragment of the state transition table with the PMC and the idea of state assignment is presented in Fig. 4.

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—$(S_p, S_r) = 1$.

The state transition table may include two transitions from the present state S_p to the next states S_a and S_b for the inputs X_u and X_w. The symbolic implicants, referring to the present state S_p, belong to the same transition function δ_i. Such a situation is interpreted as a secondary merging condition $\{S_p\}_{\delta_i,X_u,X_w}^{S_a,S_b}$. The SMC $\{S_p\}_{\delta_i,X_u,X_w}^{S_a,S_b}$ 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 the state encoding process with the mechanism of SMC arising is shown in Fig. 5.

The basic difficulty in effective term use, 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 effects are unforeseen. The elements of two-level minimization or methods of counting the number of implicants (as the effect of the minimization process) have to be included in the process of state assignment. It is easy to search primary merging conditions, but secondary merging conditions appear only in the state assignment process and come from the distribution of implicants among the single functions.

The **Implicants Distribution Table (IDT)** is a 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 1 on the i-th position of the code. When the PMC or the SMC is fulfilled, -1 is written into column corresponding to the function δ_i, for

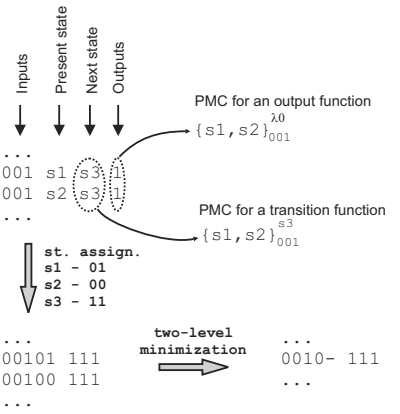

Fig. 4. Fragment of the state transition table with the PMC and the idea of state assignment.

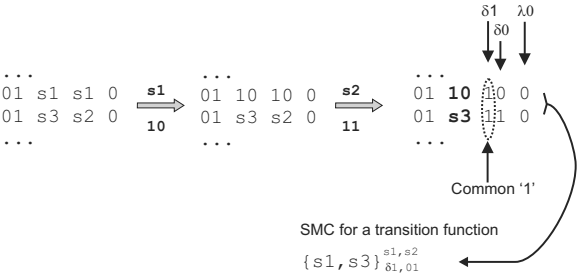

Fig. 5. Mechanism of SMC forming during the process of state assignment.

which two implicants are merged. An example of an IDT is shown in Fig. 7.

The aim of the method of state assignment is minimizing the number of PAL-cells. The minimization is limited to the transition function only.

Algorithm *mb*
(**m**inimization of the number of **b**locks)

1. Calculate the number K of bits of codes: $K = \lceil \log_2 \mathrm{card}(S) \rceil$, where $\mathrm{card}(S)$ is the number of states ($\mathrm{card}(S) > 1$), $\lceil a \rceil$ is a minimum integer but not less than a.

2. Specify the PMCs of the transition function.

3. Assign the state with the greater weight η^{S_i} with the zero code ($\mu = 0$). If there is more than one state that satisfies the condition, choose the state S_j which can satisfy most PMCs $\{S_i, S_r\}_X^{S_j}$.

4. $\mu := 1$.

5. Choose the state with the largest 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 $\{S_i, S_r\}_{\delta_j, X}^{S_a, S_b}$.

6. If none of the μ-range codes is free, then $\mu := \mu + 1$.

7. Assign to the chosen state S_i a free code μ-range. If there is more than one possibility, the sort key is as follows:

   (a) PAL-cell incrementation is the smallest,

   (b) the sum of all Δ_{δ_i} is the smallest.

   (PAL-cell incrementation and the sum of all Δ_{δ_i} are calculated after making allowance for every satisfied merging condition.)

8. Refresh the IDT.

9. Revise the secondary merging conditions.

10. Cancel the satisfied or excluded primary and secondary merging conditions.

11. If not all states have been encoded, then return to the point 5.

12. End.

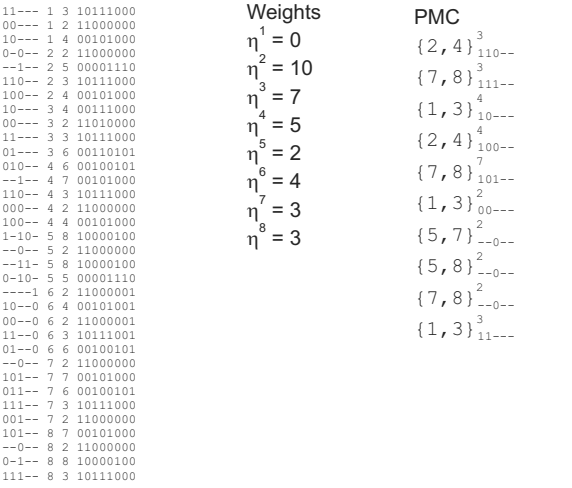

Fig. 6. State transition table of the *ex6* FSM with weights of states and PMCs.

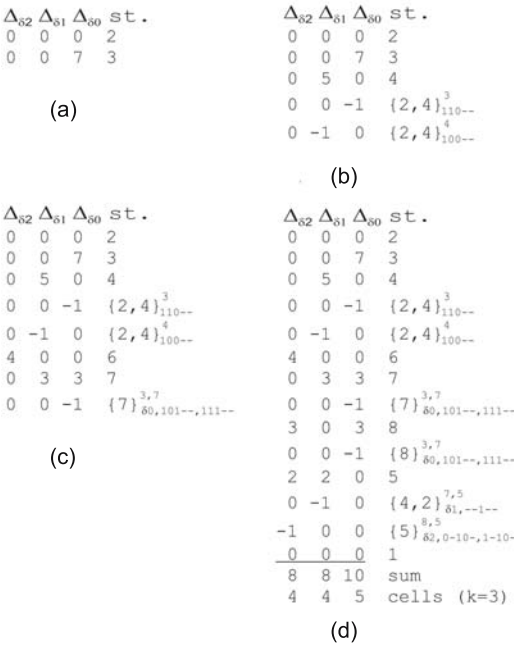

Fig. 7. IDT of the *ex6* FSM in different stages of the state assignment process.

Example 2. Consider the example of the automaton *ex6* (MCNC, 1991). The STT is given in Fig. 6 with weights of states and PMCs.

The weight η^1 of the state 1 is the biggest, the state is assigned 000. Three rows of the part of the IDT presented in Fig. 7(a) correspond to the numbers of implicants, which are equal to weights of the states. According to the IDT, the weights of states are written into the column δ_i for which there is 1 on the i-th position of the code. The state 3 is assigned 001.

The continuation of state assignment is presented in Fig. 7(b). The state 4 is assigned 010. The PMC $\{2,4\}^{3}_{110--}$ and $\{2,4\}^{4}_{100--}$ are fulfilled, so -1 is written into the ITD for the column corresponding to the function δ_0 and δ_1.

Next, the state 6 is assigned 100 and the state 7 – 011 (Fig. 7(c)). It should be noticed that the states 3 and 7 have common logic high on the position corresponding to the function δ_0. Because there are two transitions from the state 7 for the inputs $101--$ and $111--$ ($\{7\}^{3,7}_{\delta_0,101--,111--}$ is fulfilled), Δ_{δ_0} is also decremented.

Encoding results in 26 implicants and implementation uses thirteen 3-AND cells (Fig. 7(d)). ♦

3.2. PAL-oriented multi-level optimization. Two basic stages of PAL-oriented multi-level optimization include

- exchange of codes,
- optimization based on the graph of outputs.

3.2.1. Exchange of codes. The algorithm of the state assignment presented in section 3.1 is oriented on the PAL-cell minimization. Searching the shared multi-output implicants is ineffective. Exchange of codes can be carried out after the state assignment process to increase the effectiveness of the method based on the graph of output (Section 3.2.2).

First of all, the state S_i with a weight exceeding one ($\eta^{S_i} > 1$) is searched. The state is assigned a code of the first range ($\mu = 1$). Transition to the state is realized thanks to $\sigma_{S_i} = \lceil(\eta^{S_i} - k)/(k-1)\rceil$ logic cells. If there exists state S_i, then state S_j with greatest weight ($\max(\eta^{S_j})$ and $\eta^{S_j} > 1$) is searched. States S_i and S_j must have 1 on common position.

As a next step, an IDT is revised. The weight of state S_i is changed to 1. The reason is the feedback from common block (shared multi-output implicants) to the interconnect matrix. Every merging condition in an IDT is revised. If after exchanging of codes the total number of logic cells is smaller than before exchanging, then the exchange is fixed. Of course to the number of logic cells after exchanging procedure is added σ^{S_i}. The procedure is repeated until there are unchecked states with assigned codes of the 1-range.

Algorithm *ec*
(**e**xchange of **c**odes)

1. Choose the state S_i.
2. If there exists the state S_i, then search the state S_j.
3. Check the IDT for exchanging codes of the states S_i and S_j.

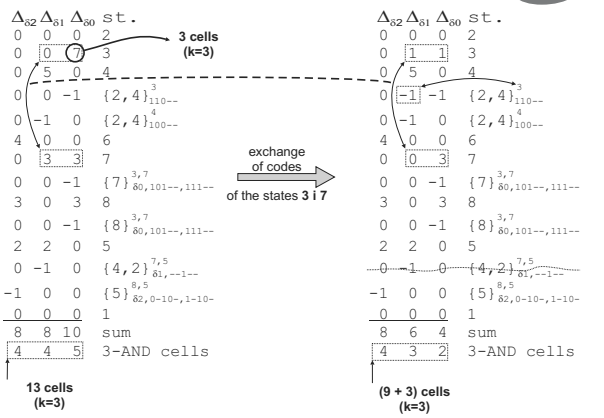

Fig. 8. Example of applying the exchange of codes procedure.

4. Exchange the codes if the realisation requires fewer logic cells than before exchanging.
5. If not all states have been checked, then go to 1.
6. End.

The procedure of exchanging codes is presented in the following example.

Example 3. On the left hand of Fig. 8 there is an IDT of the *ex6* automata after state assignment presented in Fig. 7 the state 3 is chosen (as first in the exchange of codes procedure). The code of the state is 001 and the weight of the state is 7. A transition to the state requires three 3-AND cells ($k = 3$). As next, the state 7 is chosen. The states 3 and 7 have 1 in the common position (δ_0—code 011) and the weight of the state 7 is greater than 1. The new weight of the state 3 is 1 because feedback from common block uses one term. A PMC $\{2,4\}^{3}_{110--}$ must be revised. A -1 is written in the columns Δ_{δ_0} and Δ_{δ_1}. The procedure of the exchange of codes makes the SMC $\{4,2\}^{7,5}_{\delta_1,--1--}$ unsatisfied. The number of cells required to implement the transition function after the exchange of codes is 9 and 3 to implement a common block (shared multi-output implicants). ♦

3.2.2. Optimization based on graph of outputs. The set of multi-output minterms of the Boolean function $f\colon B^n \to B^m$ serves as a starting point for PAL-oriented multi-level optimization based on the graph of outputs. The minimized form of multi-output functions $f\colon B^n \to B^m$ can be described by a set of multi-output implicants. The input part of multi-output implicants is composed of $\{0,1,-\}$, while an output part is composed of $\{0,1\}$ (De Micheli, 1994). Let y be an m component output vector that is associated with the output part of the multi-output implicant. The number of the same y_i vectors that constitute the set of multi-output implicants will be called

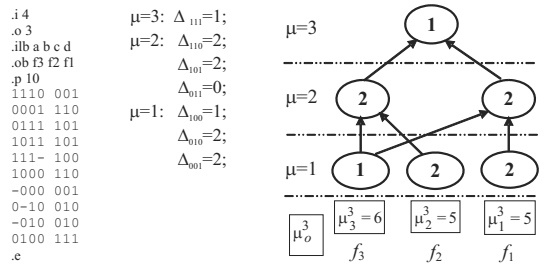

Fig. 9. Representation of the minimized function $f\colon B^4 \to B^3$ by means of the graph of outputs.

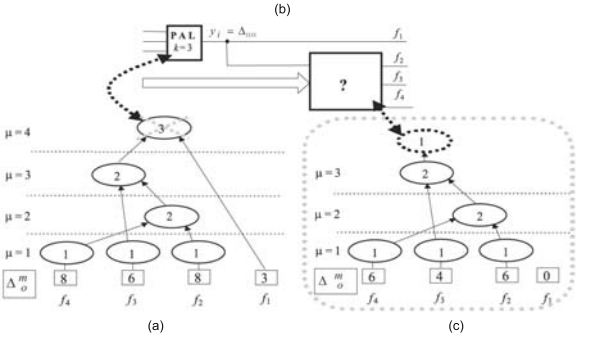

Fig. 10. Graph of the function $f\colon B^4 \to B^3$ (a), implementation of the implicants defined by the fourth-range node (b), graph after reduction (c).

the discriminant Δ_y. Let $\mu(\Delta_y)$ (range of the Δ_y discriminant) be the number of 1 components included in the y vector. Let us assume that $G\left\langle Y, \bar{U}\right\rangle$ is the directed graph, where Y is the set of graph nodes Δ_y while $\bar{U}$ is a set of graph edges. Each graph edge connects Δ_{y_s}, Δ_{y_r} nodes of the graph, if the code distance of the y_s, y_r vectors is 1, and $\mu(\Delta_{ys}) + 1 = \mu(\Delta_{yr})$. The reduced graph, presented in Fig. 9, is obtained by means of eliminating from the primary graph nodes for which $\Delta_y = 0$. To simplify, the nodes of the graph contain only decimal value of discriminants. Every node of the range one that is related to the implicants of the o-th output of the m-output function can be associated with the decimal value Δ_o^m. The discriminant Δ_o^m is equal to the sum of discriminants included in nodes covered by all the paths starting from this node and ending in nodes of the upper ranges (Fig. 9). The number of PAL-based blocks, which are necessary for the implementation of the multi-output function, can be calculated based on values of the discriminants Δ_o^m. This number is equal to

$$\sigma_f = \sum_{o=1}^{m} \left(\left\lceil \frac{\Delta_{m_o} - k}{k-1} \right\rceil + 1 \right)$$

and, for most cases, greater than σ_f^1.

Nodes of the graph correspond to the number of multi-output implicants. For example, when a node of the μ-th range belongs to the graph and for that node $\Delta_y = k$, the implementation of the k implicants constituting common resources of the μ functions is possible within this one block. Selecting the node leads to the transformation of the graph and a corresponding reduction of the Δ_o^m coefficients (Fig. 10). As a result, the selection of a certain node introduces feedback loops marked on the graph by means of the dashed line. The feedback loop is shown on the reduced graph in Fig. 10 after implementation of the fourth-range nodes.

Let ${}^i\Delta_y$ be the discriminants that correspond to the node which is chosen during the i-th step of the algorithm of multi-level optimization of the multi-output function. If

$${}^i\sigma_f - {}^{i+1}\sigma_f > \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1,$$

then the implementation of the group of implicants which correspond to the selected node ${}^i\Delta_y$ may lead to the minimization of the number of the employed PAL-based blocks composed of k terms. The selection of the node ${}^i\Delta_y$ affects the $\mu({}^i\Delta_y)$ discriminants ${}^i\Delta_o^m$. After having the discriminants re-ordered in such a way that the selected node affects the consecutive ${}^i\Delta_j^{\mu({}^i\Delta_y)}$ discriminants, the condition for the minimization of PAL-based logic blocks can be shown in the following form:

$$S^i - S^{i+1} > \left\lceil \frac{{}^i\Delta_y - k}{k-1} \right\rceil + 1$$

where

$$S^i = \sum_{j=1}^{\mu({}^i\Delta_y)} \left(\left\lceil \frac{{}^i\Delta_j^{\mu({}^i\Delta_y)} - k}{k-1} \right\rceil + 1 \right),$$

$$S^{i+1} = \sum_{j=1}^{\mu({}^i\Delta_y)} \left(\left\lceil \frac{{}^{i+1}\Delta_j^{\mu({}^i\Delta_y)} - k}{k-1} \right\rceil + 1 \right).$$

Let $r_j^{\mu({}^i\Delta_y)}$ be number calculated from the congruence

$${}^i\Delta_j^{\mu({}^i\Delta_y)} - 1 \equiv r_j^{\mu({}^i\Delta_y)} (\mod (k-1)),$$

where $j = 1, 2, \ldots, \mu({}^i\Delta_y)$.

The essence of the theorem presented in (Kania, 2004) consists in the selection of nodes (multi-output implicants) that can be shared by several single-output functions. The theorem serves as a background to draw up an algorithm for multi-level optimization of multi-output functions for PAL-based logic blocks. As the number of logic blocks σ_f is generally greater than the value of σ_f^1, the essence of the proposed algorithm focuses on analyzing the graph nodes and searching for nodes that are associated with possible large groups of common implicants.

Let us assume that the graph of outputs describes multi-output implicants that are accomplished by means

of ${}^{i}\sigma_f$ PAL-based logic blocks with k terms. Selection, at the i-th step of iteration, the ${}^{i}\Delta_y$ node of the graph implies the utilization of ${}^{i}\gamma = \lceil ({}^{i}\Delta_y - k)/(k-1) \rceil + 1$ PAL-based logic blocks. This leads to the reduction of the graph of outputs. The graph after the reduction describes the implicants that are covered by ${}^{i+1}\sigma_f$ PAL-based logic blocks. The higher the value of the expression ${}^{1}\sigma_f - ({}^{i+1}\sigma_f + {}^{i}\gamma)$, the better the selection of the node in question. The rules for the selection of the node can be deduced directly from the theorem (Kania, 2004) and can be listed as follows

1. Firstly, at the very beginning, one has to choose the ${}^{i}\Delta_y$ node, for which $\mu({}^{i}\Delta_y) = \max$.
2. From the nodes of the same range, further selection must be carried out depending on values of discriminants:
   (a) *if there exist nodes for which* ${}^{i}\Delta_y \geq k$—the node, for which the discriminant ${}^{i}\Delta_y = \max$.
   (b) *if values of all the discriminants are lower than* k—the node for which within the set of remainders $R = \left\{ r_j^{\mu({}^{i}\Delta_y)}; j \in \langle 1, \mu({}^{i}\Delta_y) \rangle \right\}$ there exists a maximum number of remainders $r_x^{\mu({}^{i}\Delta_y)}$ that meet the condition $0 < r_x^{\mu({}^{i}\Delta_y)} <^{i} \Delta_y < k$.

The above rules are the basis for the algorithm of PAL-oriented multi-level optimization.

Example 4. Let us consider the example of the automaton *ex6* (MCNC, 1991). Only five functions of the output block will be considered for simplicity. The *so_min.pla* file after classical single-output minimization dedicated for PAL-based implementation (Espresso-Dso) is presented in Fig. 11(a).

Classical implementation of the *so_min.pla* file (Fig. 11(a)) requires 15 logic PAL-based blocks which include three terms and contain three logic levels.

Let us consider the multi-output function $f\colon B^8 \rightarrow B^5$, which, after multi-output minimization (Espresso), can be depicted in the *mo_min.pla* file (Fig. 11(b)). The reduced graph of outputs, associated with *mo_min.pla*, is shown in Fig. 12.

Direct realization of implicants with PAL-based logic blocks that contain three terms each (by means of the classical method, after minimization the multi-output function) needs

$${}^{0}\sigma_f = \sum_{o=1}^{5} \left(\left\lceil \frac{{}^{0}\Delta_o^5 - k}{k-1} \right\rceil + 1 \right) = 4+5+3+1+2 = 15$$

PAL-based logic blocks ($k = 3$).

The node Δ_{01110} does not meet conditions of minimization (Kania, 2004), hence the realization of the associated implicant by means of a separate PAL-based logic

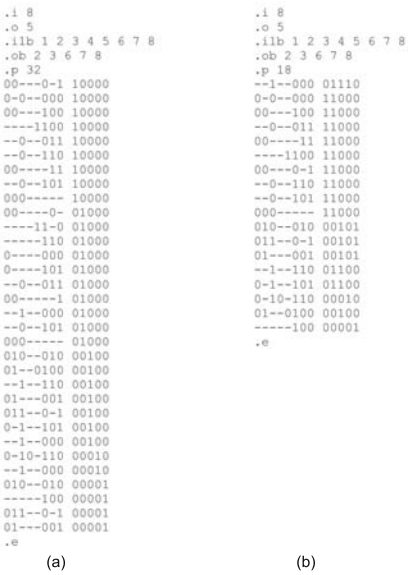

Fig. 11. (a) *so_min.pla* file of the *ex6* FSM after single-output minimization, (b) *mo_min.pla* after multi-output minimization.

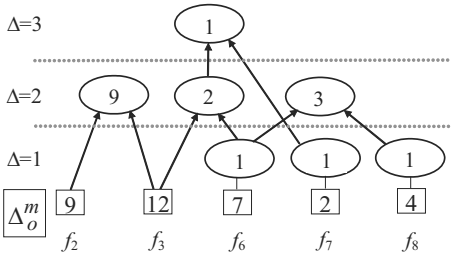

Fig. 12. Reduced graph of outputs.

block is not justified (makes no sense). A much better solution consists in the transformation of the minimized function illustrated by the graph by means of the so-called node splitting.

The node Δ_{01110} corresponds to the first implicant of the *mo_min.pla* file (Fig. 11(b)) ----1----000 01110. The implicant can be replaced with other ones by appropriate modification of the output part. The best splitting can be found by analysing the graph of outputs. The output parts should correspond to the output parts of the implicants that are associated with the nodes of lower ranges which are connected with the node altered. Figure 13 illustrates the graph of outputs after splitting the node Δ_{01110}.

During the next steps of the proposed algorithm, the implicants associated with the nodes $\Delta_{11000} = 9$, $\Delta_{01100} = 3$, $\Delta_{00101} = 3$ are realized. This step involves six PAL-based logic blocks that contain three products (terms) (${}^{1}\gamma = 6$) and lead to significant lowering of the number of blocks that are necessary for direct realization

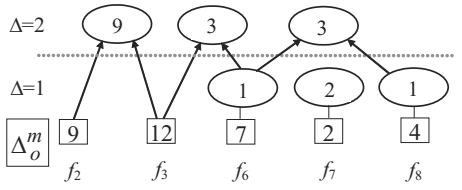

Fig. 13. Graph of outputs after splitting the third-range node.

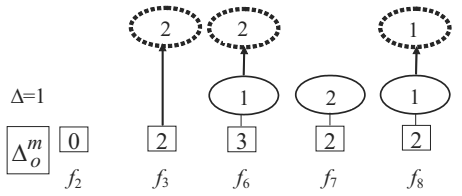

Fig. 14. Graph of outputs with feedback loop description.

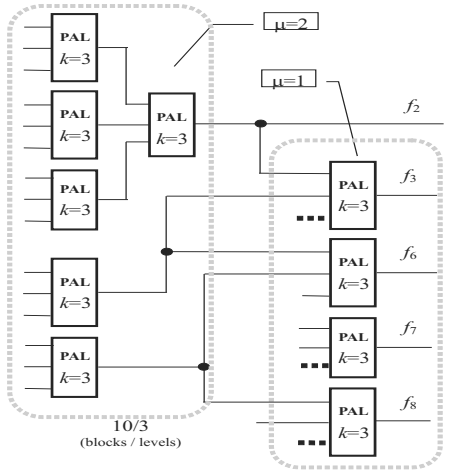

Fig. 15. Results of optimization based on the graph of outputs.

of implicants depicted by the reduced graph:

$$
\begin{aligned}
{}^1\sigma_f + {}^1\gamma &= \sum_{o=1}^{5} \left(\left\lceil \frac{{}^1\Delta_o^5 - k}{k-1} \right\rceil + 1 \right) \\
&= 0 + 1 + 1 + 1 + 1 + 6 = 10 \leq {}^0\sigma_f .
\end{aligned}
$$

That graph is obtained after removing the nodes $\Delta_{11000} = 9$, $\Delta_{01100} = 3$, $\Delta_{00101} = 3$ and introducing an additional node which is connected with these nodes and represents the feedback loop (Fig. 14).

The nodes that correspond to the first range (Fig. 14) are implemented at the last stage of the synthesis process. The final circuit representation that uses ten PAL-based logic blocks is shown in Fig. 15.

♦

3.2.3. Algorithm of FSMs synthesis for PAL-based devices. The algorithms presented in Section 3.1 make a complex strategy of FSM synthesis. The strategy is presented in the algorithm *cb*. Espresso is used as the minimization method.

Algorithm *cb*
(**c**ommon **b**locks searching)

1. An *mb* state assignment.
2. Espresso Dso; a *pla* analysis.
3. Optimization based on the graph of outputs; a *pla* analysis.
4. Exchange of codes (*ec*).
5. If there was a change in the point 4, then: Espresso and optimization based on the graph of outputs; a *pla* analysis.
6. Choosing the best (of 2, 3, 5) result.
7. End.

4. Experimental results

The method of implementing an FSM in PAL-based structures presented in the paper was compared with another approach with respect to the number of the logic blocks used and the number of logic levels. The experiments were carried out by means of NOVA (Villa and Sangiovanni-Vincentelli, 1990; Sentovich *et al.*, 1992) and the proposed methods using 40 selected benchmarks (MCNC, 1991). Experiments by NOVA were run for algorithms: the input and output (dominance) constraints (iohybrid code—ioh) and the input constraints (ihybrid code—ih). A vast majority of contemporary CPLD contains 5-AND cells. Because some terms are used to ensure some extra functions (T flip-flop, product terms expansion, etc.), experiments were run also for 3-, 4- and (6–9)-AND cells.

The results of experiments for PAL-based logic blocks with three, four and five terms are presented in Table 2. The results of experiments for (6–9)-AND cells are presented in Table 3. The numbers (a/b) show the results of synthesis performed on the benchmarks using the *mb*, the *cb*, the NOVA-ih, and the NOVA-ioh approach. The first number (a) shows the number of the employed k-product PAL-based blocks of the transition function, while the second number (b) is the number of logic levels of the transition function.

Among all examined benchmarks implemented based on PAL-based logic blocks with three terms, the proposed methods provided 17 solutions (43%) which required a smaller number of logic blocks, and 14 solutions

(35%) which demanded a greater number of logic blocks than a better solution from the Nova-ih and the Nova-ioh approach. In all other cases, the numbers of logic blocks obtained for both methods were identical.

In the set of 40 compared cases, the proposed *cb* algorithm found the best 24 solutions (60%) did not require a greater number of logic blocks with three terms than the better solution from the Nova-ih and Nova-ioh methods. Among them, 16 implementations (67%) demanded the smallest number of logic blocks and only six solutions (25%) required a greater number of logic levels. For certain benchmarks, the reduction of logic block count was significant, e.g., *dk27* (18%), *keyb* (38%), *planet* (19%), *s1488* (18%), *s1494* (17%), *s386* (13%), *styr* (19%), *tma* (23%). Significant differences can be noticed not only for the smallest values of k. The reduction of 4-terms PAL-based logic blocks is as follows *keyb* (32%), *planet* (20%), *tma* (17%), *sse* (17%), *s420* (25%), *s386* (17%).

Unfortunately, the number of logic levels does not follow the reduction of the number of logic blocks. Among the examined benchmarks implemented based on 3-terms PAL-based logic blocks, five implementations (13%) required a greater number of logic levels with respect to the better result from the NOVA-ih and the NOVA-ioh method, and four solutions (10%) needed a smaller number of logic levels. In all other cases (77%), the numbers of obtained logic levels for both methods were identical.

The analysis of the obtained experimental results is presented in the form of a yield of the logic cells $U\sigma^{\delta}$ and a yield of the logic levels $U\xi^{\delta}$ of the transition function. The yield of the logic cells $U\sigma^{\delta}$ is calculated from the equation $U\sigma^{\delta} = \left(\sum \sigma^{\delta}(A) - \sum \sigma^{\delta}(M)\right) / \sum \sigma^{\delta}(A)$, where $\sum \sigma^{\delta}(A)$ denotes the average of the whole number of transition blocks of the selected benchmarks, while the average is calculated for all tested method; $\sum \sigma^{\delta}(M)$ denotes the whole number of transition blocks of the selected benchmarks for the selected encoding method.

The yield of the logic levels $U\xi^{\delta}$ is calculated analogically to the yield of the logic cells $U\sigma^{\delta}$. The yield of the logic cells (levels) should be interpreted as the percent of the number of the logic cells (levels) for which the selected method is better (or worse if the yield is negative) than the average.

The graphs in Figs. 16 and 17 present the yield of the logic cells and logic levels of the experimental results. First of all, it should be noticed that the yield of logic cells for the *cb* algorithm is more than 6% for the smallest logic cells ($k = 3$). Moreover, the yield of the logic cells is greater than 3% for 4-AND cells and about 2% for 5-AND cells. The *mb* method gives results comparable with NOVA.

The graphs presented in Figs. 18 and 19 concern the experimental results of "small" automata. As "small" automata, the authors of the paper found automata with the

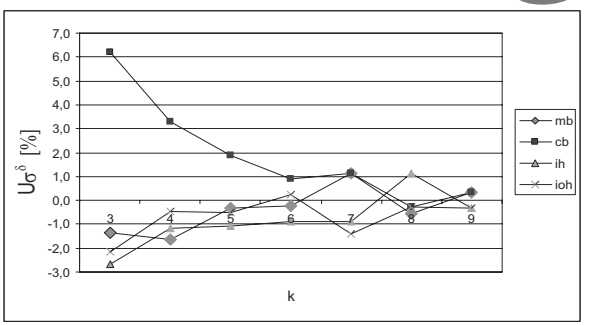

Fig. 16. Yield of the logic cells.

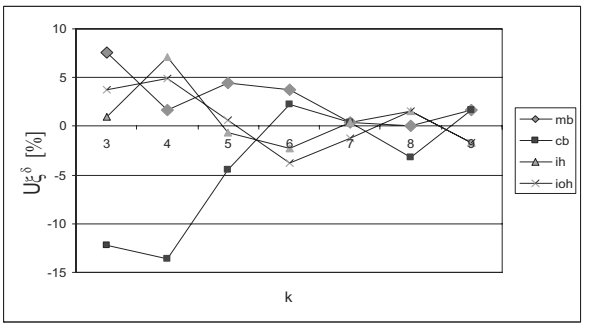

Fig. 17. Yield of the logic levels.

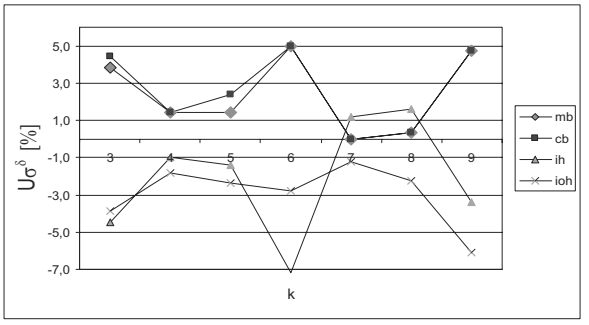

Fig. 18. Yield of the logic cells of "small" automata.

number of terms (in the *kiss2* format) less than 60. Those results are especially interesting. Both methods, *mb* and *cb*, gave results better than NOVA in relation to the yield of the logic cells and comparable results in relation to the yield of the logic levels. The *cb* method does not improve significantly the results received after state assignment process carried out by means of the *mb* algorithm.

The heuristic methods of state assignment and PAL-oriented multi-level optimization (proposed in the paper) can be implemented as quick computer algorithms. PAL-oriented multi-level optimization ensures searching out a solution which is not worse with respect to the number of blocks than the one generated by synthesis without multi-level optimization. The optimization is carried out at the expense of slightly increased calculation time. Computational complexity of the proposed algorithms is less than

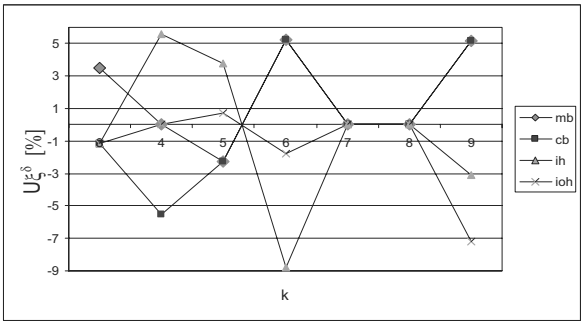

Fig. 19. Yield of the logic levels of "small" automata.

that of NOVA algorithms. Example synthesis times for selected benchmarks are presented in Table 1. Differences are especially noticeable for big benchmarks. Sometimes, synthesis time exceeded one day for the NOVA-ih algorithm. The presented computational complexity comparison is only approximate. The experiments were run for different conditions (different computers and operating systems), so the results should be treated warily.

The obtained results have also been compared to vendor tools (Czerwiński and Kulisz, 2009). Commercial vendor-independent systems use low level netlist formats to export structures resulting from logical synthesis to vendor systems. This approach is secure, because there is little chance that the low level structure will be interfered with by implementation tools. The method is, however, not universal, because netlist formats are vendor specific. Using this approach requires thus equipping software with procedures or plugins responsible for converting formats and preparing data specific for implementation tools. This is acceptable for commercial companies but difficult for academic research teams, as it requires much "scientifically worthless" extra work. The purpose of the presented approach (Czerwiński and Kulisz, 2009) is to use a special style of HDL description as the intermediate format. The proposed format is universal and portable, and it is much more comprehensible to a human than low level netlists. The method was verified for main vendor synthesis systems and CPLD circuits. The experimental results proved that the method is secure, i.e., the investigated implementation tools did not deteriorate the results of logic synthesis.

Table 1. Synthesis times for selected benchmarks.

Benchmark	cb	ih	ioh
planet	3 [s]	>1800 [s]	20 [s]
train11	<1 [s]	28 [s]	4 [s]

5. Conclusions

The synthesis of finite state machines dedicated for PAL-based CPLDs was presented in the paper. The essence of the method lies in incorporating two-level minimization into the synthesis process. All steps of the proposed method are directed to efficient implementation of FSMs into CPLDs with PAL-based logic blocks containing a predefined number of product terms. The synthesis of FSMs consists of PAL-oriented state assignment using primary and secondary merging conditions and PAL oriented multi-level optimization based on the graph of outputs. PAL-oriented multi-level optimization can also be used for the optimization of combinational circuits.

Subsequent steps of the synthesis process are adapted to logical resources of PAL-based CPLDs. The elements of two-level minimization are included in the process of state assignment. The procedure of coding word changing and PAL-oriented multi-level optimization make the method of FSM synthesis competitive. Adjustments of state assignment to logical resources characteristic for a PAL-based logic block enables significant improvement of synthesis effectiveness in relation to other approaches. Moreover, it is worth noting that the achieved FSMs are self-correcting. The reason for this is the usage of coding words with all inactive output levels.

The proposed method is an alternative to FSMs classical technology mapping based on classical coding (binary, one-hot, Gray) with elements of two level minimization of individual single-output functions and other academic methods like YEDI, NOVA, etc. The results of experiments presented in the paper prove that the proposed synthesis method is especially attractive for CPLD structures consisting of small PAL-based logic blocks. The results of synthesis of "small" automata are especially interesting. Both the proposed methods gave results better than NOVA with respect to the number of logic levels and the number of PAL-based blocks. For all cases (not only a contrived benchmark), the solutions generated by the proposed method were minimal with respect to the number of the PAL-based blocks used.

References

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

Barkalov, A., Titarenko, L. and Chmielewski, 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): 565–575.

Chattopadhyay, S. (2001). Low power state assignment and flipflop selection for finite state machine synthesis—A genetic algorithmic approach, *IEE Proceedings—Computers and Digital Techniques* **148**(45): 147–151.

Chyży, M. and Kłosiński, W. (2002). Evolutionary algorithm for state assignment of finite state machines, *Proceedings of*

Table 2. Direct comparison of experimental results.

	$k=3$				$k=4$				$k=5$			
b-mark	mb	cb	ih	ioh	mb	cb	ih	ioh	mb	cb	ih	ioh
bbara[2]	19/3	17/3	12/2	15/3	12/2	12/2	8/2	10/2	10/2	9/2	7/2	9/2
bbsse[1]	14/2	14/2	17/3	18/3	10/2	10/2	12/2	12/2	9/2	9/2	9/2	10/2
bbtas[1]	4/2	4/2	5/2	5/2	4/2	4/2	4/2	4/2	3/1	3/1	3/1	4/2
beecnt[1]	3/1	3/1	3/1	6/2	3/1	3/1	3/1	5/2	3/1	3/1	3/1	4/2
cse[2]	27/3	22/4	25/3	32/3	19/3	18/3	18/2	22/3	14/2	14/2	14/2	18/2
dk14[1]	12/3	12/3	9/2	12/3	8/2	8/2	6/2	8/2	6/2	6/2	6/2	7/2
dk15[1]	3/2	3/2	3/2	5/2	3/2	3/2	2/1	3/2	2/1	2/1	2/1	3/2
dk16[2]	40/3	40/3	37/3	36/3	28/3	28/3	25/3	24/2	23/2	23/2	20/2	19/2
dk17[1]	8/2	8/2	7/2	8/2	6/2	6/2	6/2	6/2	5/2	5/2	4/2	5/2
dk27[1]	3/1	3/1	5/2	4/2	3/1	3/1	3/1	4/2	3/1	3/1	3/1	3/1
dk512[1]	9/2	9/2	10/2	11/2	8/2	8/2	7/2	8/2	8/2	7/2	6/2	7/2
ex1[2]	31/3	30/4	26/3	29/3	22/3	22/4	19/2	20/2	19/2	19/2	14/2	16/2
ex4[1]	6/2	6/2	14/2	6/2	6/2	6/3	10/2	5/2	5/2	5/2	8/2	4/1
ex6[1]	10/2	10/2	11/3	11/3	7/2	7/2	8/2	8/2	6/2	6/2	6/2	6/2
keyb[2]	31/3	31/4	50/4	77/4	23/3	23/3	34/3	52/3	16/2	16/2	27/3	39/3
lion[1]	3/2	3/2	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
lion9[1]	8/2	8/2	5/2	4/1	7/2	7/2	4/1	4/1	6/2	6/2	4/1	4/1
mark1[1]	11/2	11/2	9/2	10/2	6/2	6/2	6/2	6/2	5/2	5/2	5/2	6/2
mc[1]	2/1	2/1	3/2	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
opus[1]	9/2	9/3	10/2	9/2	7/2	7/2	7/2	7/2	5/2	5/2	7/2	6/2
planet[2]	56/3	54/5	67/3	72/4	39/3	37/4	46/3	50/3	30/2	30/2	35/3	36/3
pma[2]	33/3	31/3	33/3	40/3	24/3	23/3	23/2	28/3	18/2	18/3	18/2	22/2
s1[2]	50/3	48/5	60/4	38/3	36/3	34/4	41/3	26/3	29/2	28/4	31/3	20/2
s1488[2]	53/3	50/5	61/4	66/3	39/3	38/4	43/3	45/3	26/2	26/2	31/3	34/3
s1494[2]	57/3	52/5	66/3	63/4	40/3	40/3	45/3	45/3	30/2	30/2	35/3	32/3
s208[2]	10/2	10/2	10/2	12/2	7/2	7/2	8/2	9/2	5/1	5/1	6/2	8/2
s27[1]	6/2	6/2	8/2	5/2	5/2	5/2	5/2	3/1	4/2	4/2	5/2	3/1
s386[2]	14/2	14/2	18/3	16/2	10/2	10/2	12/2	12/2	8/2	8/2	10/2	8/2
s420[2]	9/2	9/2	10/2	12/2	6/2	6/2	8/2	9/2	6/2	6/2	6/2	8/2
s510[2]	57/3	52/5	56/4	58/3	38/3	36/4	40/3	39/3	28/2	28/2	30/3	29/2
s820[2]	45/3	41/4	37/3	33/3	33/3	32/4	25/3	22/3	21/2	21/2	19/2	18/2
s832[2]	48/3	44/4	36/3	32/3	32/3	30/4	25/3	22/3	27/3	27/3	19/2	17/2
sand[2]	64/4	59/5	66/4	58/3	44/3	44/5	46/3	40/3	32/3	32/3	34/3	30/3
sse[1]	16/3	15/3	17/3	18/3	10/2	10/3	12/2	12/2	8/2	8/2	9/2	10/2
styr[2]	63/4	55/5	70/4	68/4	46/3	41/4	48/3	46/3	34/3	32/4	36/3	35/3
tav[1]	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
tbk[2]	127/4	102/6	93/5	72/4	85/4	68/6	63/4	50/3	63/3	56/6	48/3	38/3
tma[1]	20/3	20/3	26/3	27/3	15/2	15/2	18/2	19/2	11/2	11/2	15/2	14/2
train11[1]	10/2	10/3	6/2	7/2	7/2	7/2	5/2	5/2	7/2	7/2	4/1	4/1
train4[1]	3/2	3/2	4/2	3/2	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
Sum	996/98	922/119	1009/105	1004/102	706/90	672/104	703/85	698/87	543/75	531/82	547/79	544/78
Sum[1]	162/41	161/43	176/43	175/43	123/36	123/38	126/34	127/36	104/34	103/34	107/32	108/33
Sum[2]	834/57	761/76	833/62	829/59	583/54	549/66	577/51	571/51	439/41	428/48	440/47	436/45

[1] the number of terms in the *kiss2* format is less than 60 ("small" automata); [2] the number of terms in the *kiss2* format is greater than 59

the Euromicro Symposium on Digital System Design, Dortmund, Germany, pp. 359–362.

Czerwiński, R. and Kania, D. (2005). State assignment for PAL-based CPLDs, *Proceedings of the 8-th Euromicro Symposium on Digital System Design, DSD2005, Porto, Portugal*, IEEE Computer Society Press, Porto, pp. 127–134.

Czerwiński, R., Kania, D. and Kulisz, J. (2006). FSMs state encoding targeting at logic level minimization, *Bulletin of the Polish Academy of Sciences* **54**(4): 479–487.

Czerwiński, R. and Kulisz, J. (2009). State machine description oriented towards effective usage of vendor-independent synthesis tool, *IFAC Workshop on Programmable Devices and Embedded Systems, PDES'2009, Roznov and Radhostem, Czech Republic*, pp. 27–32.

Czerwiński, R. (2006). *The FSMs State Assignment for PAL-Based Matrix Programmable Structures*, Ph.D. thesis, Silesian University of Technology, Gliwice, (in Polish).

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

Devadas, S. and Newton, A. R. (1991). Exact algorithms for output encoding, state assignment and four-level boolean minimization, *IEEE Transactions on Computer-Aided Design* **10**(1): 13–27.

Jóźwiak, L. and Volf, F. (1995). Efficient decomposition of assigned sequential machines and boolean functions for PLD implementations, *Proceedings of Electronic Technology Directions to the Year 2000, Adelaide, Australia*, pp. 258–266.

Table 3. Direct comparison of experimental results for (6–9)-AND cells.

	$k=6$		$k=7$		$k=8$		$k=9$	
b-mark	mb	cb	mb	cb	mb	cb	mb	cb
bbara	8/2	8/2	8/2	8/2	7/2	7/2	6/2	6/2
bbsse	7/2	7/2	6/2	6/2	6/2	6/2	4/1	4/1
bbtas	3/1	3/1	3/1	3/1	3/1	3/1	3/1	3/1
beecount	3/1	3/1	3/1	3/1	3/1	3/1	3/1	3/1
cse	12/2	12/2	9/2	9/2	8/2	8/2	8/2	8/2
dk14	6/2	6/2	6/2	6/2	5/2	5/2	4/2	4/2
dk15	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
dk16	18/2	18/2	14/2	14/2	15/2	15/2	11/2	11/2
dk17	3/1	3/1	3/1	3/1	3/1	3/1	3/1	3/1
dk27	3/1	3/1	3/1	3/1	3/1	3/1	3/1	3/1
dk512	4/1	4/1	4/1	4/1	4/1	4/1	4/1	4/1
ex1	15/2	15/2	13/2	13/2	11/2	11/2	10/2	10/2
ex4	4/1	4/1	4/1	4/1	4/1	4/1	4/1	4/1
ex6	6/2	6/2	6/2	6/2	4/2	4/2	3/1	3/1
keyb	13/2	13/2	12/2	12/2	11/2	11/2	9/2	9/2
lion	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
lion9	4/1	4/1	4/1	4/1	4/1	4/1	4/1	4/1
mark1	4/1	4/1	4/1	4/1	4/1	4/1	4/1	4/1
mc	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
opus	5/2	5/2	4/1	4/1	4/1	4/1	4/1	4/1
planet	24/2	24/2	19/2	19/2	19/2	19/2	18/2	18/2
pma	15/2	15/2	13/2	13/2	12/2	12/2	11/2	11/2
s1	22/2	22/2	19/2	19/2	17/2	17/2	16/2	16/2
s1488	24/2	24/2	22/2	22/2	17/2	17/2	17/2	17/2
s1494	25/2	25/2	19/2	19/2	19/2	19/2	16/2	16/2
s208	5/1	5/1	5/1	5/1	5/1	5/1	5/1	5/1
s27	3/1	3/1	3/1	3/1	3/1	3/1	3/1	3/1
s386	8/2	8/2	6/2	6/2	6/2	6/2	4/1	4/1
s420	5/1	5/1	5/1	5/1	5/1	5/1	5/1	5/1
s510	23/2	23/2	21/2	21/2	18/2	18/2	17/2	17/2
s820	21/2	21/2	18/2	18/2	15/2	15/2	13/2	13/2
s832	19/2	19/2	17/2	17/2	15/2	15/2	13/2	13/2
sand	27/2	27/2	22/2	22/2	20/2	20/2	17/2	17/2
sse	7/2	7/2	7/2	7/2	6/2	6/2	4/1	4/1
styr	27/2	27/2	22/2	22/2	18/2	18/2	17/2	17/2
tav	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
tbk	54/3	49/4	41/3	41/3	37/3	36/5	34/3	34/3
tma	10/2	10/2	9/2	9/2	8/2	8/2	6/2	6/2
train11	4/1	4/1	4/1	4/1	4/1	4/1	4/1	4/1
train4	2/1	2/1	2/1	2/1	2/1	2/1	2/1	2/1
sum	451/64	446/65	388/63	388/63	353/63	352/65	317/59	317/59

Kania, D. (2003). An efficient approach to synthesis of multi-output boolean functions on PAL-based devices, *IEE Proceedings on Computer and Digital Techniques* **150**(3): 143–149.

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

MCNC (1991). LGSynth'91 benchmarks, Collaborative Benchmarking Laboratory, Department of Computer Science at North Carolina State University, Raleigh, NC, `http://www.cbl.ncsu.edu/`.

Mengibar, L., Entrena, L., M.G.Lorenz and E.S.Millan (2005). Patitioned state encoding for low power in FPGAs, *Electronics Letters* **41**(17): 948–949.

Park, S., Yang, S. and Cho, S. (2000). Optimal state assignment technique for partial scan designs, *Electronics Letters* **36**(18): 1527–1529.

Salauyou, V., Klimowicz, A., Grzes, T., Dimitrova-Grekow, T. and Bulatowa, I. (2006). Experimental Studies of Finite State Machines Synthesis Methods Implemented in Package ZUBR, *Pomiary, Automatyka, Kontrola* **52**(6 bis): 44–46, (in Polish).

Sentovich, E., Singh, K., Moon, C., Savoj, H., Brayton, R. and Sangiovanni-Vincentelli, A. (1992). SIS: A system for sequential circuit synthesis, *Technical report*, University of California, Berkeley, CA.

Sharma, K. (1998). *Programmable Logic Handbook, PLDs, CPLDs, & FPGAs*, McGraw-Hill, New York, NY.

Villa, T., Kam, T., Brayton, R. and Sangiovanni-Vincentelli, A. (1997). *Synthesis of Finite State Machines: Logic Optimization*, Kluwer Academic Publishers, Boston, MA.

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

Yang, S. and Ciesielski, M. (1991). Optimum and suboptimum algorithms for input encoding and its relationship to logic minimization, *IEEE Transactions on Computer-Aided Design* **10**(1): 4–12.

Robert Czerwiński received his M.Sc. and Ph.D. degrees in technical sciences from the Faculty of Automatics, Electronics and Computer Science, Silesian University of Technology, Poland, in 2001 and 2006, respectively. At present he works at the Institute of Electronics of the same university as an assistant professor. His research interests include programmable logic devices, logic synthesis and optimization.

Dariusz Kania received his M.Sc. and Ph.D. degrees from the Silesian University of Technology, Gliwice, Poland, in 1989 and 1995, respectively. He has worked as an assistant lecturer (1989–1995) and an assistant professor (1995–2004). Since 2006 he has been a professor at the Silesian University of Technology. His main interests and research areas include programmable devices and systems, logic synthesis dedicated to a wide range of programmable logic devices (CPLD, FPGA), and the implementation of digital circuits.

Received: 25 September 2008
Revised: 14 March 2009
Re-revised: 28 April 2009