BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 58, No. 4, 2010
DOI: 10.2478/v10175-010-0067-6

Synthesis method of high speed finite state machines

R. CZERWIŃSKI* and D. KANIA

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

Abstract. The paper is concerned with the problem of state assignment and logic optimization of high speed finite state machines. The method is designed for PAL-based CPLDs implementations. Determining the number of logic levels of the transition function before the state encoding process, and keeping the constraints during the process is the main problem at hand. A number of coding bits, as well as codes for the states, are adjusted to achieve a machine with a determined number of logic levels. Elements of two-level minimization are taken into consideration in the state assignment. The proposed optimization method is based on utilizing tri-state buffers, thus enabling achievement of a one-logic-level output block.

Key words: state assignment, Finite State Machines (FSM), Programmable Array Logic (PAL), Complex Programmable Logic Devices (CPLD), logic optimization, tri-state buffer.

1. Introduction

Two most popular families of programmable logic devices are FPGAs (Field Programmable Gate Arrays) and CPLDs (Complex Programmable Logic Devices). FPGAs are developed definitely faster than CPLDs, however, the structure of CPLDs is more efficient than LUT-based structures of the FPGAs [1–3].

A large majority of CPLDs are built of a simple cell matrix and a programmable interconnect array (PIA) – see Fig. 1.

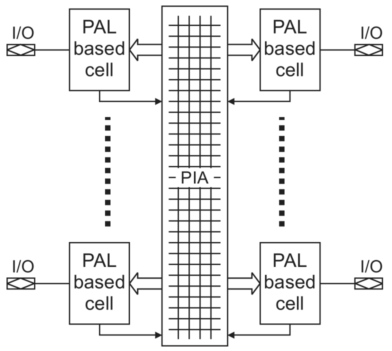

Fig. 1. Typical CPLD structure

The core of most CPLDs is PAL-based cell. The generalized structure of the PAL-based cell is shown in Fig. 2.

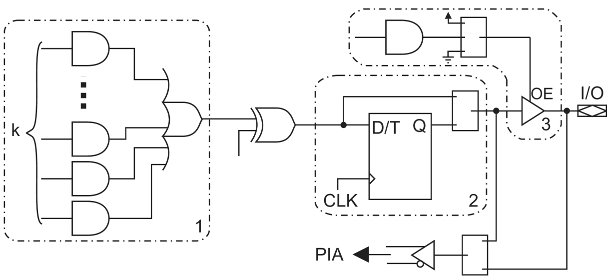

Fig. 2. Generalized structure of PAL-based cell

PAL-based cell contains a programmable-AND/fixed-OR structure (1), which can implement logic up to k product terms. In most cases $k = 5$ (Altera: MAX3000A; Xilinx: XC9500, MAX7000; Lattice: ispXPLD5000; Atmel: ATF1500). The output of an AND-gate cannot 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). An intrinsic part of the cell is the tri-state buffer (3). Generally, an OE input can be driven by a combinational circuit (usually an AND-gate) or is connected to logic 'high' or 'low'. The built-in tri-state buffer enables the expansion process, among other things.

Logic blocks contained in CPLD structures usually feature additional logic resources that can facilitate product term expansion. These resources include parallel expanders, folded NAND feedback lines, often referred to as shared expanders, logic allocators. The expanders enable unequal distribution of product terms between cells, and extending the number of products available for one function beyond the limit of k terms contained in one PAL block. Anyway they can only move the limit to a greater value, and they do not provide feasibility of implementation for every function. Additional expansion of the number of terms is thus necessary.

The proposed logic synthesis process consists of two main procedures:

- PAL-oriented state assignment,
- PAL-oriented two-level optimization of output block.

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

The state assignment is basic and the most important stage of FSMs synthesis. Despite the fact that methods considered as optimal were developed [4, 5], the works on the synthesis for CPLDs are still continued [6, 7]. To be certain, there are also many aims for the optimization, like reducing the

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

power consumption of automata [8, 9] or synthesis for testability [10, 11].

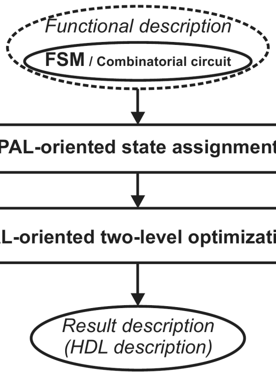

Fig. 3. Logic synthesis of FSMs for PAL-based devices

This paper is structured as follows. Section 2 focuses on the basic definitions. The essence of the method is presented in Sec. 3. Experimental results are reported in Sec. 4. The paper concludes with a summary in Sec. 5.

2. Theoretical background

2.1. The automata theory. The mathematical model of a sequential circuit is a Finite State Machine (FSM), which is a five-tuple: $\{\mathbb{X}, \mathbb{Y}, \mathbb{S}, \delta, \lambda\}$, where: $\mathbb{X}$ is a finite input alphabet, $\mathbb{Y}$ is a finite output alphabet, $\mathbb{S}$ is a finite set of states, δ is the transition function, and λ is the output function. The transition function of an FSM determines the next state of the automata (S^+), and is the mapping $\delta\colon \mathbb{X}\times\mathbb{S}\to\mathbb{S}$. The output function is associated with each transition: $\lambda\colon \mathbb{X}\times\mathbb{S}\to\mathbb{Y}$ or with each state: $\lambda\colon \mathbb{S}\to\mathbb{Y}$. The structure of the FSM is presented in Fig. 4.

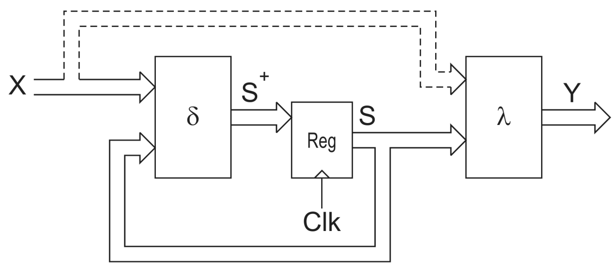

Fig. 4. Structure of an FSM

Internal states of an FSM are given mostly symbolic values. The goal of the state assignment is to assign to every state a binary representation. The minimum number of code bits K can be calculated from Eq. (1):

$$K = \lceil \log_2 card(S) \rceil \,, \tag{1}$$

where $\lceil a \rceil$ is a minimum integer not less than a, and $card(S)$ is the number of internal 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). The rows of a STT are called symbolic implicants. A state transition graph, with a corresponding STT, is presented in Fig. 5.

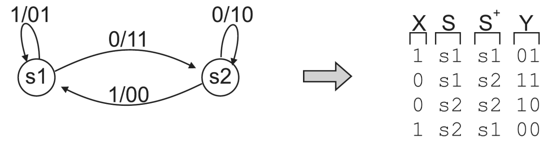

Fig. 5. State transition graph and corresponding STT

2.2. Basic definitions. A multi-output implicant of a function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ is a pair of row vectors of dimension n and m called an input and an output part, respectively. The input vector items are taken from the set $\{0, 1, -\}$ and represents a product of literals. The output part has entries in the set $\{0, 1\}$. For each output component, a 1 implies a true or don't care value of the function in correspondence with an input part. A multi-output Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}^m$ may be represented as a collection of m single-output functions $f_i\colon \mathbb{B}^n \to \mathbb{B}^1$ $(i = 0, \ldots, m-1)$.

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; whereas an output part of the same corresponds to a next state and the primary output. Generally, the δ and λ functions are multi-output functions, so let δ_i be i^{th} bit of the transition function and λ_j be j^{th} bit of the output function.

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 an STT.

Let Δ_{f_i} be a number of implicants of the function f_i, e.g. Δ_{δ_i} is 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 in which they differ in.

Let the ξ_δ be the number of logic levels of the transition block.

Let $card(Y)$ be the cardinality of the Y set.

In order to simplify and increase a clarity of figures, symbols of the PAL-based cell will be used in the paper, instead of drawing a full cell. The PAL-based cell symbols used in the paper are presented in Fig. 6.

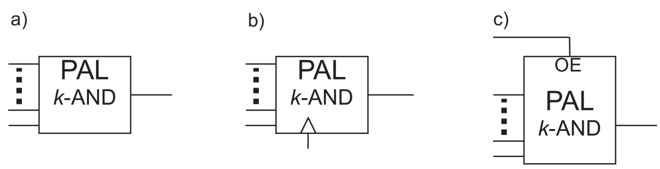

Fig. 6. Symbols of PAL-based cell: a) with bypassed flip-flop, b) with flip-flop, c) with tri-state buffer driven by AND-gate

Let σ_f be a number of PAL-based cells of the implementation of the function f.

Let ξ_f be a number of cascaded PAL-based cells in the longest signal path from the inputs to the outputs.

Product term expansion using feedbacks to a PIA causes an addition of extra logic levels to the structure. Let the structure of PAL-based cell (like in Fig. 6) insert to the path a delay defined as a one-logic-level. Hereafter in this paper, we will interpret the term "one-logic-level", "ξ_f-logic-level" as the number of cascaded PAL-based cells in the longest signal path from the inputs to the outputs in the concerned circuit. The exception to this rule will be the terms "two-level minimization", and "two-level optimization". These terms are well established in the literature, and we will be used in their traditional meaning, i.e. "two-level" = two levels of logic gates.

2.3. Introduction to a state assignment. Because a coded STT is a collection of multi-output implicants, 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'. Before going ahead, one more thing should be noticed: the state with the greatest weight should be assigned the code with all ($\mu = 0$) bits logic low. This is because none of the single transition functions includes implicants corresponding to transition to the state. Elements that refer to the weights of states have been proposed in [12].

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. Of course the number of terms may be reduced as the effect of two-level minimization. The main goal of the state assignment process should be to assign states with codes conveniently situated for implicant merger. 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 and secondary merging conditions enable the algorithm to include elements of two-level minimization into the process of the state assignment.

2.4. 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 the 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 the figure is $\nu(s1, s2) = 2$. Implicants cannot be merged.

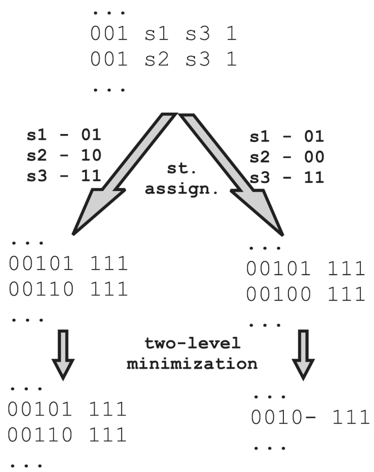

Fig. 7. Fragment of an example FSM with two types of state assignment

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.

A Primary merging condition for the output function ($\{S_p, S_r\}_X^{\lambda_i}$) is defined as 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.

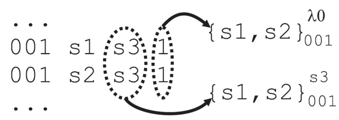

Fig. 8. A part of STT with primary merging conditions

2.5. 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 $i = 0, \ldots, m-1$. 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 split to single-output implicants (because product terms of 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 δ_2. It is of course possible because the distance between codes of the states $s1$ and $s3$ equals one.

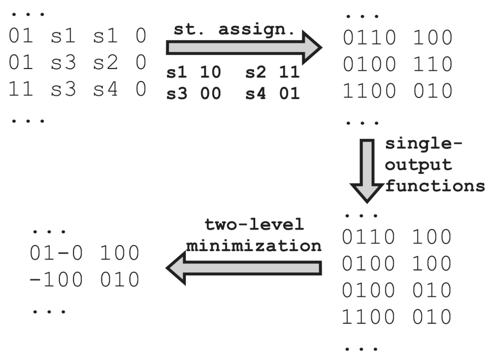

Fig. 9. A part of an STT before and after the state assignment process

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 δ_1.

Definition 2. A Secondary Merging Condition (SMC) $\{S_p, S_r\}^{S_a,S_b}_{\delta_i,X}$ 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 $\{S_p, S_r\}^{S_a,S_b}_{\delta_i,X}$, the states S_p and S_r have to be assigned binary codes with the distance between them equal to one – $\nu(S_p, S_r) = 1$.

One more secondary merging condition ($\{S_p\}^{S_a,S_b}_{\delta_i,X_u,x_w}$) can be defined as 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 the state encoding process is shown in Fig. 10. The mechanism of the SMC arising is also presented.

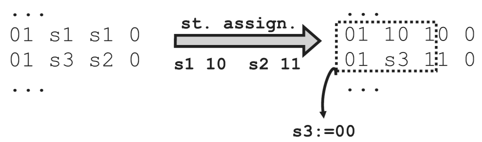

Fig. 10. The mechanism of the SMCs forming, during the process of state assignment

2.6. The implicants distribution table. The basic difficulty of 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 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 3. The Implicants Distribution Table (IDT) T is a table divided into columns, corresponding to the weights Δ_{δ_i} of the single functions δ_i. Every row of the table corresponds to the number of implicants which is equal to the weights of the states. These are written into those columns Δ_{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 the column corresponding to the function δ_i, for which two implicants are merged.

2.7. Elements of two-level optimization. Classical logic synthesis of combinational circuits implemented in great majority of vendor tools consists of two steps. First a two-level minimization is applied separately to every single-output function. Then, implementation of the minimized functions in PAL-based blocks, containing a predefined number of product terms, is performed. If the number of implicants Δ_{f_i}, representing a function after minimization, is greater than the number of product terms k, available in a logic block, a greater number of logic blocks has to be utilized to implement the function. The classical product term expansion method consists in utilizing feedback lines to build a multi-level cascaded structure, which increases propagation delays significantly.

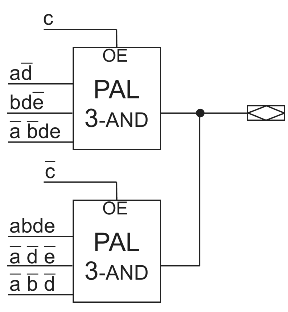

Fig. 11. The example of product term expansion exploiting tri-state output buffers

Consider the function $y = ac\overline{d} + ab\overline{c}de + bcd\overline{e} + \overline{a}\overline{b}cde + \overline{a}\overline{c}\overline{d}\overline{e} + \overline{a}\overline{b}\overline{c}\overline{d}$. This function can be implemented using two PAL-based logic blocks with 3 terms per output and tri-state output buffers (Fig. 11).

Product term expansion that exploits tri-state output buffers seems to be the most attractive solution, as it does not lead to expansion of logic levels. This idea is the basis of two-level optimization.

3. The design method

3.1. PAL-oriented state assignment. The main cost of expanding product terms using feedbacks to a PIA is a reduction of the system speed, caused by the added extra logic levels to the structure. The number of logic levels of fast automata must be as few as possible. The logic level extraction problem is solved in the presented approach.

If the number of logic levels of the transition function ξ_δ were known before the state assignment, the number of code bits and codes as such could be adjusted to achieve the number of logic levels. The question is: is it possible to estimate the minimum number of logic levels of the transition block, for which a realization is possible? The answer is yes. It can be determined from the Eq. (2).

$$\xi_\delta = \begin{cases} 1 & \text{if } \eta^{S_i} < k \\ \lceil \lg_k \eta^{S_i} \rceil & \text{if } \eta^{S_i} \geq k \end{cases} \tag{2}$$

where η^{S_i} is the greatest but one weight (unless there are two, or more states with the same greatest weight). Why the greatest, but only one? Because the state with the greatest weight is assigned the zero code, so none of the functions has implicants corresponding to transitions to the state. Of course the logic level number of the transition block ξ_δ is equal to the number of cells used in the longest path.

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

The algorithm *ml* (state assignment oriented on the minimization of logic levels):

1. Calculate the number K of bits of coding word (Eq. (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 $\{s_i, s_r\}_x^{s_j}$.
4. $\mu := 1$.
5. Calculate the number ξ_δ of logic levels of the transition function (Eq. (2)).
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 Δ_{δ_i} is the smallest,

   (a) and b) are calculated after making allowance for every satisfied merging condition)
9. If exists δ_i: $\xi_{\delta_i} > \xi_\delta$, then:
   (a) cancel the last assignment,
   (b) $K := K + 1$,
   (c) $\mu := 1$,
   (d) supplement 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. End.

Let's consider an example. The STT of the example FSM with weights of states are given in Fig. 12 (*kiss2*). The coding length K is 4. It has been assumed that $k = 5$. The number of logic levels is determined on the basis of weight of the state 4 and of course equals one.

```
.i 6
.o 9
.p 21
.s 14
1----- 1  3  110000000
1----- 3  2  000000000
1----- 2  5  001000000
1----- 5  7  000000000
10---- 7  7  000000000
11---- 7  11 100110000
1----- 11 12 100100000
1-1--- 12 8  000001100
1-0--- 12 8  000000100
1-0--- 8  3  110000000
1-10-- 8  3  110000000
1-11-- 8  4  110000000
1---1- 4  13 000000010
1---0- 4  13 000000000
1----- 13 14 001000010
1----- 14 6  000000000
10---- 6  6  000000000
11---- 6  9  100110000
1----- 9  10 100100000
1----1 10 3  110000101
1----0 10 4  110000100
```

Weights

$\eta^1 = 0$
$\eta^2 = 1$
$\eta^3 = 4$
$\eta^4 = 2$
$\eta^5 = 1$
$\eta^6 = 2$
$\eta^7 = 2$
$\eta^8 = 2$
$\eta^9 = 1$
$\eta^{10} = 1$
$\eta^{11} = 1$
$\eta^{12} = 1$
$\eta^{13} = 2$
$\eta^{14} = 1$

Fig. 12. An example function with weights

The state 3 is assigned first of all because the weight η^3 is the greatest. According to the algorithm the state 3 is assigned

0000. Next, states 6, 7, 13 and 4 are assigned respectively 0001, 0010, 0100 and 1000. According to the definition 3, 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 in Fig. 13 part of the IDT correspond to the numbers of implicants, which are equal to weights of the states, that is 2.

Because none of the 1-range codes are free, so $\mu := 2$. First, the state 8 is assigned 0011, and then the state 11 is assigned 1010. The SMC $\{7\}^{7,11}_{\delta 1,11----,10----}$ is fulfilled, so a -1 is written into ITD for the column corresponding to the function δ_1 (Δ_{δ_1} is decremented). A similar situation occurs after encoding the state 9. Next, states 2, 10, 12 and 14 are encoded.

```
η^δ3 η^δ2 η^δ1 η^δ0  st.
0    0    0    0     3
               2     6
          2          7
     2               13
2                    4
          2    2     8
1         1          11
         -1          {7}^{7,11}_{δ1,11----,10----}
     1         1     9
              -1     {6}^{6,9}_{δ0,11----,10----}
     1    1          2
1              1     10
1    1               12
     1    1    1     14
              -1     {14,9}^{6,10}_{δ0,1-----}
         -1          {3,13}^{2,14}_{δ1,1-----}
    -1               {3,13}^{2,14}_{δ2,1-----}
5    5    5    5     sum
```

Fig. 13. State assignment process

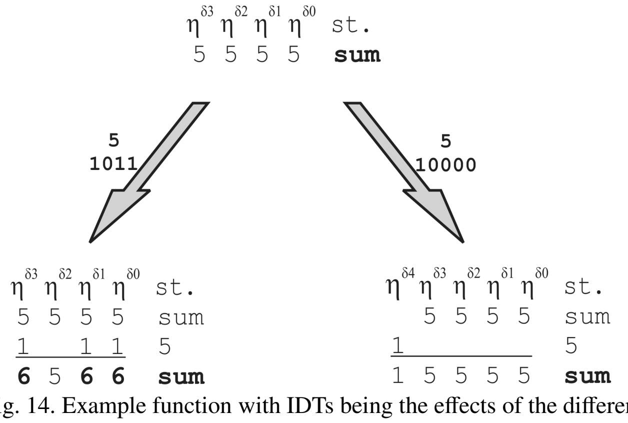

Fig. 14. Example function with IDTs being the effects of the different state assignment

A starting point to assign state 5 is an IDT form Fig. 13. The state 5 should be assigned one of the free code, e.g. 1011. Using this code, as well as any other free code, makes the structure of the transition function two-logic-level. In this situation, an additional bit of code must be used. A state 5 is assigned with code 10000. The number of the logic levels still remains one, and used logic cells are smaller than in the case of using 4-bit codes. Codes, which are already used, are supplemented with 0 on the MSB position. An assignment of the state 5 is presented in Fig. 14.

3.2. PAL-oriented two-level optimization. The concept of two-level optimization of FSM's output block lies in the background of the original method of product term expansion utilizing tri-state terminals.

The set of multi-output implicants of a Boolean output function $f\colon \mathbb{B}^n \to \{0,1,-\}^m$ serves as the starting point for a two-level optimization. The two-level optimization consists of a two-level splitting minimization, PAL-oriented term partitioning, and PAL mapping. The optimization process starts with the two-level splitting minimization. Then partitioning of the individual minimized functions is performed. As a result of the two procedures, the set of implicants of a Boolean function is divided into subsets with cardinality less than the number of terms available in one PAL-based cell.

The objective of the classical two-level minimization is to reduce both the number of products in the Boolean formula representing a function, and the number of literals in a product. Because of a limited number of multi-input terms available in PAL-based cell, the primary goal of the two-level splitting minimization is to reduce the number of products. Reduction of literals is non-essential. The idea of the two-level splitting minimization is presented in Fig. 15.

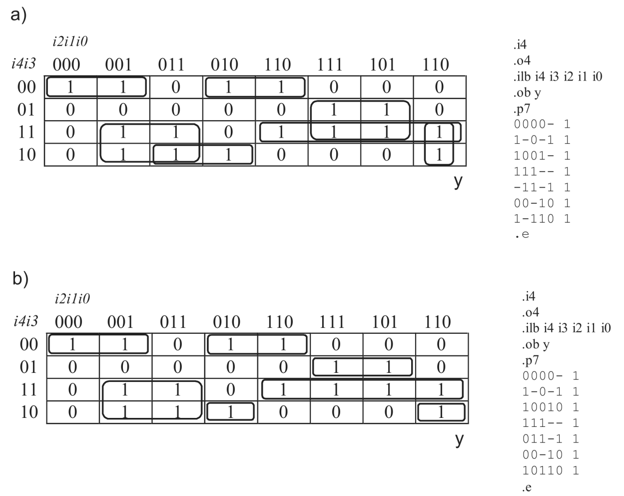

Fig. 15. The essence of the two-level splitting minimization: a) results of the classical two-level minimization, b) results of the two-level splitting minimization

The process of two-level splitting minimization starts from classical two-level minimization using the Espresso algorithm. Then, modification of individual minimized functions is executed in succession by means of an implicant splitting procedure.

Let an implicant $A_{ys} = a_{(n-1)s}, \ldots, a_{1s}, a_{0s}$ covers 2^{r_s} minterms of single-output function $y = f(i_{n-1}, \ldots, i_1, i_0)$ that form the set of minterms $\mathbb{I}_s$, whereas implicant $A_{yt} = a_{(n-1)t}, \ldots, a_{1t}, a_{0t}$ covers 2^{r_t} minterms that form the set of minterms $\mathbb{I}_t$. Where the two implicants $A_{ys} = a_{(n-1)s}, \ldots, a_{1s}, a_{0s}$ and $A_{yt} = a_{(n-1)t}, \ldots, a_{1t}, a_{0t}$ are half-mutual-covering, it turned out that there exists the possibility to modify one of the two implicants, while the set $\mathbb{I}_{s\cup t} = \mathbb{I}_s \cup \mathbb{I}_t$ would not be changed. The search for half-mutual-covering pairs of implicants consists in analysis of ordered pairs $\langle a_{is}, a_{it}\rangle$, where $i = 0, \ldots, n-1$. Let the A_{ys} implicant contains not fewer components, such as $a_{is} = \{-\}$, than the A_{yt} implicant. The implicants A_{ys} and A_{yt} are half-mutual-covering ones in the case, when among components of set of ordered pairs, there are pairs that belong to the set $\{\langle 0,0\rangle, \langle 1,1\rangle, \langle -,-\rangle, \langle -,0\rangle, \langle -,1\rangle\}$ and there is only one pair $\langle a^*_{is}, a^*_{it}\rangle$ that belongs to the set $\{\langle 0,-\rangle, \langle 1,-\rangle\}$. Modification, that would not change the set of minterms $\mathbb{I}_{s\cup t} = \mathbb{I}_s \cup \mathbb{I}_t$ covered both by A_{ys} and A_{yt} implicants consists in replacing of the a^*_{it} component following the rule: if $a^*_{is} = 1$, then $a^*_{it} := 0$, whereas if $a^*_{is} = 0$ then $a^*_{it} := 1$.

For instance, if set of implicants is

$$\left\{\begin{matrix} 0 & - & - & - \\ - & - & 0 & 0 \end{matrix}\right\}_{i3i2i1i0}$$

then, after carrying out modification,

$$\left\{\begin{matrix} 0 & - & - & - \\ 1 & - & 0 & 0 \end{matrix}\right\}_{i3i2i1i0}.$$

The presented rule serves as a basis for an implicant splitting procedure that carries out a search for all the half-mutual-covering implicants and converts one of these belonging to that pair. As a result of that process all the pairs of half-mutual-covering implicants are to be modified, which leads to obtaining a set of implicants which consisting of a minimized number of components $a_i = \{-\}$.

The consecutive steps of splitting implicants for one function of *5xp1* are presented in Fig. 16.

After two-level splitting minimization, PAL-oriented term partitioning is executed. The objective of the PAL-oriented term partitioning procedure is to subdivide the set of implicants into subsets, for which cardinality is less or equal to the number of terms (k) available in PAL-based cell.

Let us consider the function from Fig. 15. Let us assume that the function is to be realized by means of the PAL-based logic cells, each of them containing 3 terms. Having completed the two-level splitting minimization, we obtained the result shown in the left-hand column of Table 1. Then we attempt to find the partition of y function implicants into such two subsets Y_1 and Y_2, that cardinality of the Y_1 set is less or equal to the number of terms (k) included in PAL-based logic cells ($card(Y_1) \leq k$), while ($card(Y_2) = \min$). The theoretical background of PAL-based partitioning is presented in [13].

In the example $i4$ implied partition of implicants into two subsets Y_1 and Y_2, for which $card(Y_1) = 3$ and $card(Y_2) = 4$ is presented in the next column of Table 1. The first one characterizes the y_1 function that is active for $i4 = 0$ (Table 1, column 2), while the second one is related to the function y_2 being active for the vectors $i4 = 1$ (Table 1, column 3).

In the next step, the partitioning of y_2 function implicants is executed. A variable $i2$ implied partition of implicants into two subsets Y_{21} (function y_{21}) and Y_{22} (function y_{22}), for which $card(Y_{21}) = card(Y_{22}) = 2 < k$, presented in the 4^{th} and 5^{th} column of Table 1 respectively.

$$\left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & - & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ - & 1 & - & - & 0 & 1 & 1 \\ 0 & - & - & 0 & 1 & - & 0 \\ 0 & - & 0 & - & 1 & - & 0 \\ - & - & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\} \mapsto \left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & \mathbf{1} & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ - & 1 & - & - & 0 & 1 & 1 \\ 0 & - & - & 0 & 1 & \mathbf{1} & 0 \\ 0 & - & 0 & - & 1 & \mathbf{1} & 0 \\ - & - & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\} \mapsto \left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & 1 & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ - & 1 & - & - & 0 & 1 & 1 \\ 0 & \mathbf{1} & - & 0 & 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 & - & 1 & 1 & 0 \\ - & - & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\} \mapsto \left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & 1 & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ \mathbf{0} & 1 & - & - & 0 & 1 & 1 \\ 0 & 1 & - & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & - & 1 & 1 & 0 \\ \mathbf{0} & - & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\} \mapsto \left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & 1 & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ 0 & 1 & - & - & 0 & 1 & 1 \\ 0 & 1 & - & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & - & 1 & 1 & 0 \\ 0 & \mathbf{0} & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\} \mapsto \left\{\begin{matrix} - & - & - & - & 1 & 0 & - \\ 0 & 0 & - & - & 1 & 1 & 0 \\ 1 & - & - & - & 0 & 1 & 1 \\ 0 & 1 & - & - & 0 & 1 & 1 \\ 0 & 1 & - & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & \mathbf{1} & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{matrix}\right\}$$

Fig. 16. Consecutive steps of splitting procedure (*only input part of implicants*)

Table 1
Partitioning of *y.pla* file implied by the variable $i4$

y.pla	y_1 active, if $i4 = 0$	y_2 active, if $i4 = 1$	y_{21} active, if $i4i2 = 10$	y_{22} active, if $i4i2 = 11$
.i5	.i5	.i5	.i5	.i5
.o1	.o1	.o1	.o1	.o1
.ilb i4,i3,i2,i1,i0	.ilb i4,i3,i2,i1,i0	.ilb i4,i3,i2,i1,i0	.ilb i4,i3,i2,i1,i0	.ilb i4,i3,i2,i1,i0
.ob y	.ob y1	.ob y2	.ob y21	.ob y22
.p 7	.p 3	.p 4	.p 2	.p 2
`0000- 1`	`0000- 1`	`1-0-1 1`	`1-0-1 1`	`111-- 1`
`1-0-1 1`	`011-1 1`	`10010 1`	`10010 1`	`10110 1`
`10010 1`	`00-10 1`	`111-- 1`	.e	.e
`111-- 1`	.e	`10110 1`		
`011-1 1`		.e	OE	OE
`00-10 1`	OE		`1-0--`	`1-1--`
`10110 1`	`0----`	OE		
.e		`1----`		

The PAL-oriented partitioning scheme and PAL mapping of the y function using blocks consisting of three terms are presented in Fig. 17.

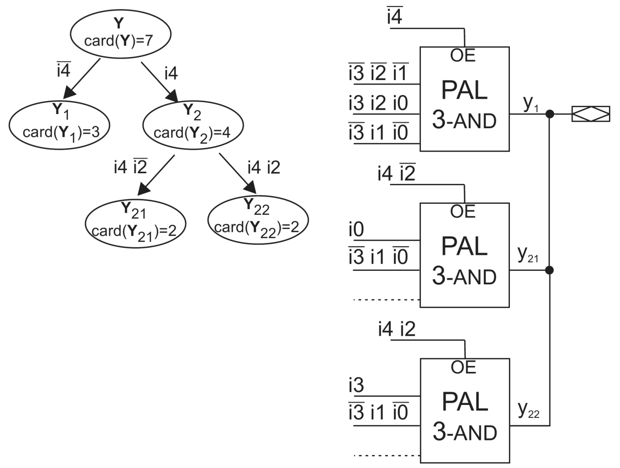

Fig. 17. Results of PAL-oriented partitioning and PAL mapping

The two-level optimization is especially attractive with respect to dynamic parameters. The algorithms discussed above can be used as independent FSM synthesis methods, improving the dynamic properties of final solutions.

An implementation of the *ex4* automaton after state assignment and optimization is presented in Fig. 18.

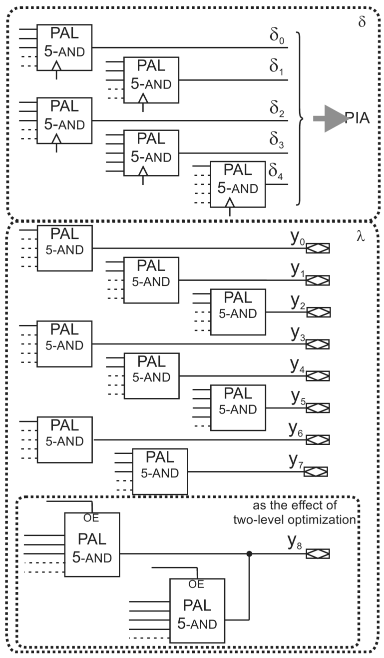

Fig. 18. Implementation of the *ex4* automaton

4. Experimental results

The experiments were carried out by means of:

- JEDI [14]: the input dominant algorithm (**i**), the output dominant algorithm (**o**) and the coupled dominant algorithm (**c**);
- NOVA [4]: the input and output (dominance) constraints (iohybrid_code – **ioh**), the input constraints (ihybrid_code – **ih**) and the input constraints (iexact_code – **ie**);
- the presented ml-algorithm (**ml**) and two-level optimization (**ml+o**).

Experiments were carried out using some selected benchmarks [15].

4.1. Analysis. Experimental results are presented in form of graphs. Two conceptions of graphs were applied:

- yield of the logic cells $U\sigma^f$ and yield of the logic levels $U\xi^f$,
- direct comparison of selected benchmarks.

The yield of the logic cells $U\sigma^f$ is calculated from the equation:

$$U\sigma^f = \frac{(\sum \sigma^f)_A - (\sum \sigma^f)_M}{(\sum \sigma^f)_A} * 100\%, \tag{3}$$

where $(\sum \sigma^f)_A$ denotes the average of cells of implementation for the function f for all the benchmarks, while the average is calculated for all tested methods; $(\sum \sigma^f)_M$ denotes the whole number of cells of the selected benchmarks for selected encoding methods. The yield of the logic levels $U\xi^f$ is calculated analogically to the yield of the logic cells $U\sigma^f$. The yield of the logic cells (levels) should be interpreted as a percent of the number of the logic cells (levels) for which selected method is better (or worst if the yield is negative) than the average. The yield is calculated for 36 benchmarks (*bbara, bbsse, bbtas, beecount, cse, dk14, dk15, dk16, dk17, dk27, dk512, ex1, ex4, ex6, keyb, lion, lion9, mark1, mc, opus, pma, s1, s208, s27, s386, s420, s820, s832, sand, sse, styr, tav, tbk, tma, train11, train4*).

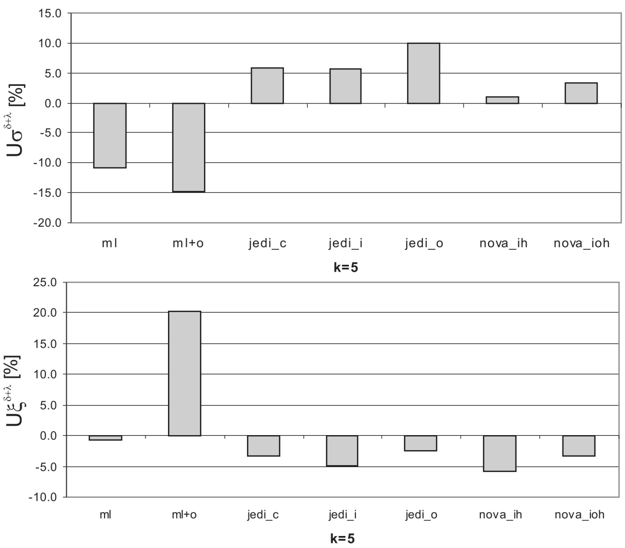

Fig. 19. Yield of the logic cells $U\sigma^{\delta+\lambda}$ and logic levels $U\xi^{\delta+\lambda}$ for different methods

As it is shown in Fig. 19, the *ml* state assignment carries out better results in comparison to NOVA and YEDI in analyzing yield of the logic levels $U\xi^{\delta+\lambda}$. The obtained results may be additionally improved as an effect of two-level optimization of the output block. The yield $U\xi^{\delta+\lambda}$ for *ml+o* method exceeds 20%. Of course, in many cases reduction of the logic levels is relevant with utilization of the excessive PAL-based cells. For the whole group of benchmarks the yield of logic cells $U\sigma^{\delta+\lambda}$ is about -10% for *ml* algorithm and nearly 15% for *ml+o* method.

Direct comparison of selected benchmarks (*bbsse, ex4, keyb, s420, sand*) is presented in Fig. 20 and Fig. 21 for the transition and output block respectively.

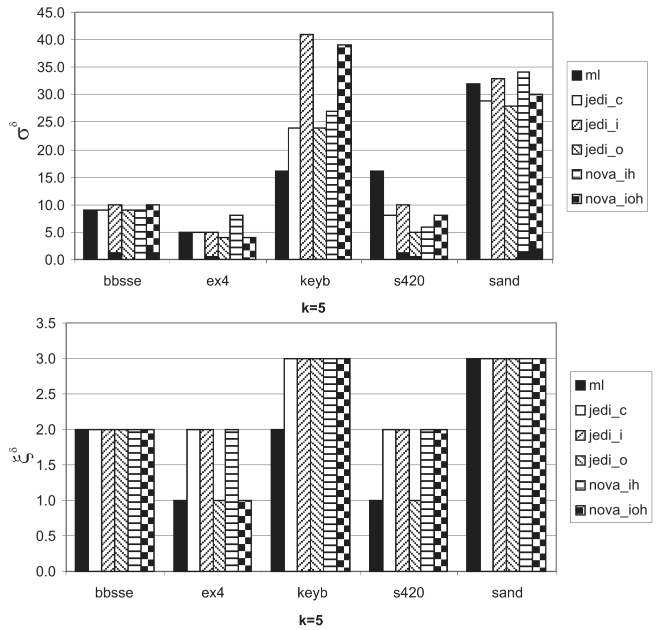

Fig. 20. Direct comparison of selected benchmarks: a transition block

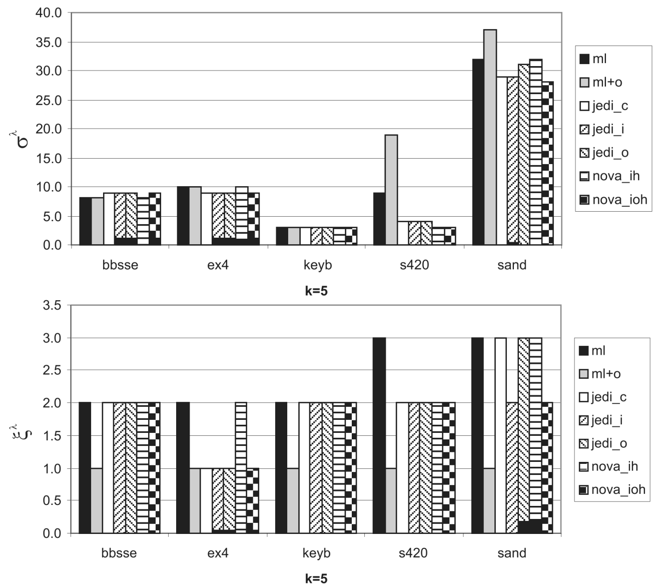

Fig. 21. Direct comparison of selected benchmarks: an output block

The logic levels ξ^{δ} obtained after state encoding by *ml* algorithm are the same (*bbsse, ex4, s420, sand*) or better (*keyb*) than results obtained by NOVA or JEDI. Moreover the results obtained for *keyb* benchmark are the best in respect to logic cells σ^{δ} in comparison to NOVA nad JEDI. However, in some cases *ml* algorithm carries out to utilize more logic cells than NOVA or JEDI (*s420*). In other cases results are comparable.

Analyzing the results of logic levels ξ^{λ} of the output block is always one-logic-level (for the whole set of 36 analyzed benchmarks). Sometimes there should be used extra logic cells to achieve one-logic-level output block as the effect of two-level optimization (*ml+o*) like for *s420* or *sand* benchmarks.

4.2. Interface to vendor tools. If thousands of experiments are to be carried out, interfacing prototype software to tools supplied by PLD vendors becomes an important issue. Software tools developed by companies or institutions independent from PLD vendors are capable of performing only the logic synthesis stage. Then, the design has to be transferred to a vendor-specific system for completing the implementation stage. This regards also academic software, developed by research teams.

The main problem in porting a design to a vendor-specific system is to find an appropriate intermediate format for the design data exchange. Commercial vendor-independent systems (e.g. Synplify, Leonardo Spectrum, Precision RTL) use low level netlists for this purpose. 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 low level netlists contain much vendor-specific and architecture-specific information. Using this approach thus requires equipping the synthesis software with procedures or plugins responsible for converting formats, and preparing data specifically for the implementation tools. This is acceptable for commercial companies, but difficult for academic research teams, as it requires much "scientifically worthless" extra work.

It was thus desirable to find alternative formats for the data exchange, possibly more universal, and using a higher level of abstraction. Here using a Hardware Description Language (HDL) seems to be the most obvious, and natural choice. Choosing the right abstraction level for the intermediate format is an important task, because vendor implementation software can change and "destroy" logical structures generated by synthesis tools.

Behavioral HDL description seems to be the design specification format most preferred for design entry nowadays. Because of its high abstraction level it allows the designer to concentrate on proper description of the desired functionality. As a textual format, following the standard of the chosen language, it is universal and portable between technologies and software tools.

A number of experiments were carried out to examine various synthesis tools, and, in particular, the effects of selecting different data exchange formats, on the quality of results. The tools were tested using the standard benchmarks [15]. The test circuits were implemented in CPLD structures.

It turned out that, if behavioural description was used as the entry format, the quality of the solutions was not good. High abstraction level in behavioural modeling gives a large degree of freedom to the software. Logical structures can easily be "spoiled" by vendor implementation programs. During the experiments it turned out, that it is possible to propose as the intermediate format a style of VHDL description, lying at a lower level of abstraction, than behavioural modeling, but still portable between software tools, and comprehensible to a human. The proposed style of VHDL modeling resembles the dataflow description commonly known in the literature. More details are reported in [13, 16].

5. Conclusions

The paper concerns the problem of high speed finite state machine designing. The automata are to be implemented in PAL-based structure, which is the core of most CPLDs.

An original method of state assignment and optimization is developed. The non-minimal state encoding is based on determining the number of the logic levels of the transition function and adjusting the number of coding bits to keep the constraints. The second idea is to keep the output block one-logic-level thanks to utilizing tri-state buffers.

REFERENCES

[1] J.H. Anderson and S.D. Brown, "Technology mapping for large complex PLDs", *Proc. Design Automation Conf.* 1, 698–703 (1998).

[2] J. Kim, S. Byun, and H. Kim, "Development of technology mapping algorithm for CPLD under time constraint", *6th Int. Conf. VLSI and CAD* 1, 411–414 (1999).

[3] A. Kaviani and S. Brown, "Technology mapping issues for an fpga with lookup tables and pla-like blocks", *Proc. 2000 ACM/SIGDA* 1, 60–66 (2000).

[4] 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).

[5] 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", *ICCD, Proc. Int. Conf. Computer Design* 1, 328–333 (1992).

[6] V. Salauyou, A. Klimowicz, T. Grześ, T. Dimitrova-Grekow, and I. Bulatowa, "Investigation of efficiency of synthesis of finite automata implemented in the ZUBR packet", *Measurements, Automatics, Control* 6, 44–46 (2006), (in Polish).

[7] A. Barkalov, L. Titarenko, and S. Chmielewski, "Reduction in the number of PAL macrocells in the circuit of a Moore FSM", *Int. J. Applied Mathematics and Computer Science* 17 (4), 565–575 (2007).

[8] S. Chattopadhyay, "Low power state assignment and flipflop selection for finite state machine synthesis – a genetic algorithmic approach", *IEE Proc. – Computers and Digital Techniques* 148 (45), 147–151 (2001).

[9] C. Cao, M. O'Nils, and B. Oelmann, "A tool for low-power synthesis of FSMs with mixed synchronous/asynchronous state memory", *IEEE Norchip Conf.* 1, 199–202 (2004).

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

[11] V. Salauyou and T. Grzes, "FSM state assignment methods for low-power design", *6th Int. Conf. Computer Information Systems and Industrial Management Applications* 1, 345–350 (2007).

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

[13] D. Kania, "The logic synthesis for programmable matrix structures of PAL type", *Scientific Notebooks of Silesian Univ. Technology* 1619, CD-ROM (2004), (in Polish).

[14] B. Lin and R. Newton, "Synthesis of multiple level logic from symbolic high-level description languages", *Proc. Int. Conf. on VLSI* 1, 187–206 (1989).

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

[16] R. Czerwiński, "The FSMs state assignment for PAL-based matrix programmable structures", *PhD Thesis*, Silesian University of Technology, Gliwice, 2006, (in Polish).