*Int. J. Appl. Math. Comput. Sci.*, 2010, Vol. 20, No. 2, 367–384
DOI: 10.2478/v10006-010-0027-1

# DECOMPOSITION–BASED LOGIC SYNTHESIS FOR PAL–BASED CPLDs

ADAM OPARA [*], DARIUSZ KANIA [**]

[*] Institute of Computer Science
Silesian University of Technology, ul. Akademicka 16, 44–100 Gliwice
e-mail: Adam.Opara@polsl.pl

[**] Institute of Electronics
Silesian University of Technology, ul. Akademicka 16, 44–100 Gliwice
e-mail: Dariusz.Kania@polsl.pl

The paper presents one concept of decomposition methods dedicated to PAL-based CPLDs. The proposed approach is an alternative to the classical one, which is based on two-level minimization of separate single-output functions. The key idea of the algorithm is to search for free blocks that could be implemented in PAL-based logic blocks containing a limited number of product terms. In order to better exploit the number of product terms, two-stage decomposition and BDD-based decomposition are to be used. In BDD-based decomposition methods, functions are represented by Reduced Ordered Binary Decision Diagrams (ROBDDs). The results of experiments prove that the proposed solution is more effective, in terms of the usage of programmable device resources, compared with the classical ones.

**Keywords:** decomposition, technology mapping, logic optimization, BDD, CPLD.

## 1. Introduction

Nowadays, Programmable Logic Devices (PLDs) are very extensively used in designing electronic digital circuits. Simple PLDs can be divided into several kinds: PAL (Programmable Array Logic), PLA (Programmable Logic Array), and PLE (Programmable Logic Element). Based on simple PLDs, there is a group of devices called Complex Programmable Logic Devices (CPLDs). Logic blocks available in CPLDs utilize a PAL-based structure (Fig. 1). Other PLDs include FPGA (Field Programmable Gate Array) devices. Due to the high complexity of these systems, efficient CAD algorithms must be developed to address their design challenges. Logic minimization and technology mapping are two important elements in this process.

The classical approach to the synthesis of PAL-based CPLD structures, implemented in many computer aided design tools, employs two-level minimization of separate functions and technological fitting to the structure of programmable logic blocks (Bolton, 1990). The Espresso algorithm is mainly used for two-level minimization of Boolean functions (Brayton *et al.*, 1984). The strategies of synthesis implemented in commercial CAD tools are designed mostly for a small group of devices produced by single manufacturers, but they do not provide effective solutions. The synthesis methods implemented in Hardware Description Language (HDL) compilers, such as the VHDL or Verilog, use a multi-level representation of functions, while the synthesis process consists in fitting a function to the technology library patterns. These methods lead to inefficient use of the available product terms in PAL-based logic.

The aim of the paper is to show a complete logic synthesis method of a multi-output function based on a new concept of decomposition adopted for a PAL-based architecture. This method is based on two-stage decomposition (Kania, 2004).

In general, a single step of functional decomposition consists of input variable partitioning into two disjoint

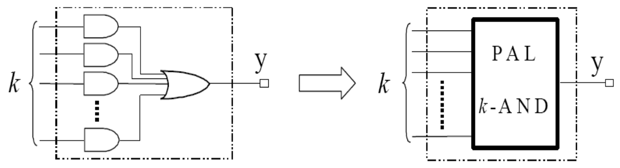


Fig. 1. Structure of a typical PAL-based CPLD block.

subsets (bound and free set), computing a symbolic function on bound set variables and input/output encoding that determines predecessor and successor functions (Murgai *et al.*, 1994).

Conventional approaches to input encoding are based on finding codes which satisfy input constraints resulting from the use of symbolic minimization. Some methods presented in (Saldanha *et al.*, 1994; Yang and Ciesielski, 1991) are targeted at two-level minimization. Proper encoding of the input and output problem is widely discussed in connection with state encoding of FSMs in (Ashar *et al.*, 1992; De Micheli, 1994). Those problems are connected with symbolic state encoding, dichotomy theory, multi-value function minimization, and output domination analysis.

For PAL-based CPLDs, more relevant are methods optimizing multi-level implementation. One of the first input encoding approaches to FPGA-based functional decomposition was presented in (Murgai *et al.*, 1994). Methods using symbolic decomposition or symbolic factorization perform encoding after decomposition. Sometimes, these methods have been applied to classical decomposition, which is not efficient for LUT-based FPGAs. Similar approaches were presented in (Burns *et al.*, 1998; Muthukumar, 2001; Muthukumar *et al.*, 2000).

The main limitation of PAL-based logic blocks is the number of multi-input product terms available in one PAL-based logic block. This results in the observation that the essence of decomposition dedicated to PAL-based structures can be reduced to two major tasks:

- Minimising the number of PAL-based logic blocks used;
- Adjusting the designed circuit to best fit the structures of PAL-based blocks.

The proposed decomposition model concerns directly the second issue. The essence of the proposed model consists in finding a design partitioning (function decomposition) which enables us to implement the free block in one PAL-based logic block containing a predefined number of product terms.

Functional decomposition backgrounds were worked out by Ashenhurst (1957) and extended by Curtis (1962). This model is the foundation of complex decomposition algorithms dedicated for the LUT-based FPGA (Burns *et al.*, 1998; Lai *et al.*, 1994; Lai *et al.*, 1996; Nowicka *et al.*, 1997; Rawski *et al.*, 2008; Scholl, 2001). The essential parts of synthesis are the partitioning and mapping of a designed circuit into a configurable logic block. There are several algorithms that utilize the decomposition in synthesis for PLA-based devices (Ciesielski and Yang, 1992; Devadas *et al.*, 1988).

A very interesting approach was presented in (Anderson and Brown, 1998; Chen *et al.*, 2002). The synthesis process developed for FPGA devices was employed, but Look-Up Table (LUT) blocks were replaced by Programmable Logic Array (PLA) cells. The optimal structure of the cells was selected after carrying out several experiments presented in (Kouloheris and Gamal, 1992). The authors prove that using small PLA cells to construct an FPGA device is more area efficient than using the LUT cell approach (Kim *et al.*, 2001; Yan, 2001). On the other hand, synthesis methods dedicated to PLA structures are well mastered and have been known for a long time (Chen and Muroga, 1988; Ciesielski and Yang, 1992; Devadas *et al.*, 1988).

The high importance of decomposition in modern logic synthesis is obvious.Functional decomposition is widely researched in the context of logic synthesis for the LUT-based FPGA and PLA-based architectures. We propose to use decomposition to minimize the area of a circuit mapped into PAL-based CPLDs.

*Is it profitable to use decomposition in the synthesis process for a PAL-based CPLD?*

This is the question this paper attempts to answer.

Only the devices with PAL-type logic blocks are considered. The novel ideas of decomposition presented here were inspired by two-stage PAL-decomposition described in (Kania, 2004; Kania *et al.*, 2005). A novel non-disjunctive PAL-decomposition based on the BDD was introduced. This decomposition model is particularly promising in the case of hardly decomposable functions.

The obtained results of experiments are (among others) compared with the classical method. This method is briefly introduced in the next subsection.

**1.1. Classical method.** The classical method of logic synthesis, dedicated to PAL-based CPLD, and implemented in the great majority of vendor tools, consists of two steps. First, two-level minimization is applied separately to every single-output function; next, the implementation of the minimized functions in PAL-based blocks, containing a predefined number of product terms, is performed. If the number of implicants $\Delta_f$, representing a function after minimization, is greater than the number of product terms $k$, available in a logic block (Fig. 1), a greater number of logic blocks has to be utilised to implement the function. The classical product term expansion method consists in utilising feedback lines to build a multi-level cascaded structure, which significantly increases propagation delays.

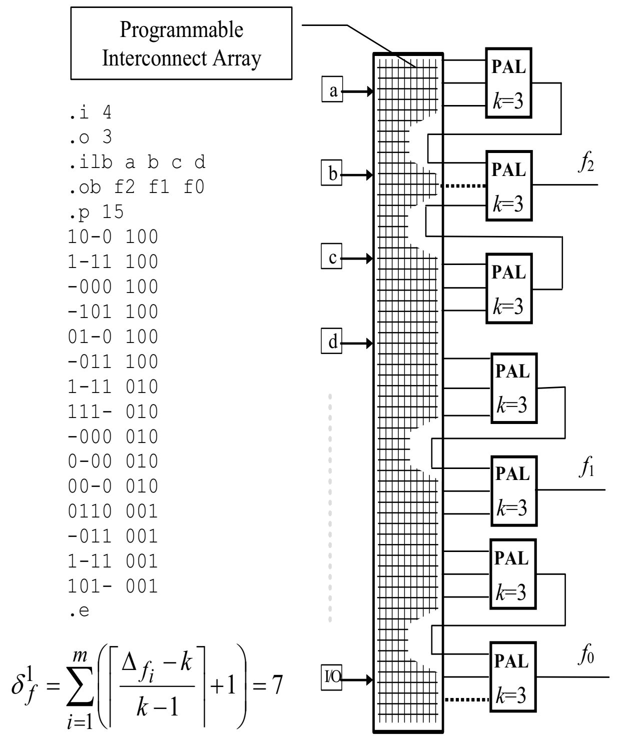
Implementing a minimized function $f$, which can be represented as a sum of $\Delta_f$ implicants, requires $\Delta_f$ PAL-based logic blocks containing $k$ product terms

$$\delta_f = \left\lceil \frac{\Delta_f - k}{k-1} \right\rceil + 1. \qquad (1)$$

Similarly, the classical implementation of a $f : B^n \rightarrow B^m$ function requires $\delta_f^1$ PAL-based logic blocks

$$\delta_f^1 = \sum_{i=1}^{m} \delta_{f_i} = \sum_{i=1}^{m} \left( \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right). \qquad (2)$$

As an example, a classical implementation, utilising PAL blocks containing three ($k = 3$) product terms of a function $f : B^4 \rightarrow B^3$, is presented (Fig. 2).



```
.i 4
.o 3
.ilb a b c d
.ob f2 f1 f0
.p 15
10-0 100
1-11 100
-000 100
-101 100
01-0 100
-011 100
1-11 010
111- 010
-000 010
0-00 010
00-0 010
0110 001
-011 001
1-11 001
101- 001
.e
```

$$\delta_f^1 = \sum_{i=1}^{m} \left( \left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right) = 7$$

Fig. 2. Classical implementation of a function $f : B^4 \rightarrow B^3$, utilising PAL-based logic blocks containing three product terms.

The purpose of this paper is to present more effective methods of function implementation in PAL-based CPLD structures. The paper is structured as follows. Section 2 presents the theoretical background and main ideas of two-stage decomposition methods dedicated to CPLDs. All steps of the proposed decomposition methods based on the BDD are given in Section 3. Experimental results are reported in Section 4. The paper closes with conclusions in Section 5.

## 2. Two-stage decomposition dedicated to PAL-based devices

The kernel of the most popular CPLDs is a PAL-based structure, which consists of a determined (in most cases, constant) number of terms connected to an output cell. The terms with the output cell are called PAL-type logic blocks.

Two-stage PAL decomposition offers more efficient logic block use than the standard two-level minimization and fitting. Adaptation to the number of terms in a PAL-based logic block is a characteristic feature of two-stage PAL decomposition. Similarly to the Ashenhurst-Curtis decomposition, the partition of a variable set into free and bound sets is of major importance (Fig. 2). The partition is chosen so that a free block can be created in one PAL-based logic block. The two-stage PAL decomposition algorithm uses a Karnaugh map as a representation of the logic function. The rows of the map are described by the values of bound variables, whilst the columns are denoted by the values of free variables. In this map, row patterns can be determined. For function $f(X)$, $X = X_f \cup X_b, X_f \cap X_b = \emptyset$, a **row pattern** is a row described by the function $g(X_b)$, which returns the value 1 for all cubes associated with the columns, for which the function $f(X)$ value is 1. In the case of the Karnaugh map depicted in Fig. 4, the first row denoted by "$g_1(X_b)$" is described by the function

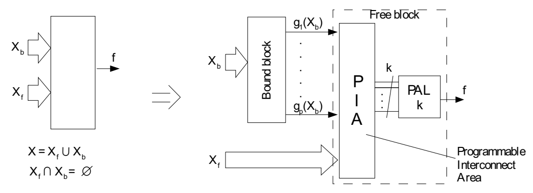


Fig. 3. Circuit partition after decomposition.

$$\begin{aligned} & f_{x0=0,x1=0,x2=0} \\ & \quad = g_1(x_3, x_4, x_5) \\ & \quad = \overline{x_3}\,\overline{x_4}x_5 + \overline{x_3}x_4\overline{x_5} + x_3x_4x_5 + x_3\overline{x_4}\,\overline{x_5}. \end{aligned}$$

There are three rows here described by this pattern:

$$\begin{aligned} & f_{x0=0,x1=0,x2=0} \\ & \quad = f_{x0=0,x1=1,x2=1} \\ & \quad = f_{x0=1,x1=1,x2=0} = g_1(x_3, x_4, x_5). \end{aligned}$$

The row pattern described by $\overline{g_1(X_b)}$ is called the **row pattern complement**. In the case of the Karnaugh map illustrated in Fig. 4, there were determined two other special cases of row patterns: a **full row** denoted by **1** and an **empty row** denoted by **0**.

The rows in the Karnaugh map can be broken down into a few groups:

- empty rows,
- full rows,
- rows associated with the same row pattern or its complement.

| $x_0x_1x_2$ \ $x_3x_4x_5$ | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 | |
|---|---|---|---|---|---|---|---|---|---|
| 000 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $\boldsymbol{g_1(X_b)}$ |
| 001 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | $\boldsymbol{g_2(X_b)}$ |
| 011 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $\boldsymbol{g_1(X_b)}$ |
| 010 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | $\boldsymbol{\overline{g_1(X_b)}}$ |
| 110 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $\boldsymbol{g_1(X_b)}$ |
| 111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 101 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | **1** |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |

$f$

$$\begin{aligned}
&X_f = \{x_0, x_1, x_2\}, \quad X_b = \{x_3, x_4, x_5\}\\
&g_1(X_b) = g_1(x_3, x_4, x_5) = \bar{x}_3\bar{x}_4x_5 + \bar{x}_3x_4\bar{x}_5\\
&\qquad\qquad + x_3x_4x_5 + x_3\bar{x}_4\bar{x}_5\\
&g_2(X_b) = g_2(x_3, x_4, x_5) = x_4x_5 + x_3x_5
\end{aligned}$$

Fig. 4. Karnaugh map with distinguished row patterns.

A **row multiplicity** of a partition matrix denoted by $\mu(X_f|X_b)$ is defined as a number of different row groups, except for the groups containing empty and full rows. It is defined by the expression (3):

$$\begin{aligned}
&f(X), \quad X = \{x_0, \dots, x_{n-1}\},\\
&X_b = \{x_q, \dots, x_{n-1}\}, \quad q < n,\\
&X_b \cup X_f = X, \quad X_b \cap X_f = \emptyset\\
&A = \big\{g(x_q, \dots, x_{n-1}) : g = f_{x_0=\beta_0,\dots,x_{q-1}=\beta_{q-1}},\\
&\qquad\qquad \beta_j \in \{0,1\}, j = 0, \dots, q-1\big\},\\
&A\text{—a row patterns set, } B\text{—a row groups set,}
\end{aligned} \tag{3}$$

$$\mu(X_f|X_b) \overset{\text{def}}{=} \Big| B : \bigwedge_{g_i, g_j \in B, g_i \neq g_j} g_i, g_j \in A,$$

$$g_i \neq \overline{g}_j, g_i \neq 0, g_i \neq 1 \Big|.$$

For the function presented in Fig. 4, the row multiplicity is $\mu(X_f|X_b) = 2$; the first group of rows contains row patterns $g_1(X_b)$ and $\overline{g_1(X_b)}$, the second—row pattern $g_2(X_b)$.

Each group of rows can be assigned a function defined as follows:

$$h_0(X_f) = \begin{cases} 1, & \text{for full rows,}\\ 0, & \text{for all other rows,}\end{cases}$$

$$h_i(X_f) = \begin{cases} 1, & \text{for rows, for which the row pattern is}\\ & \text{described by the function } g_i(X_b),\\ 0, & \text{for all other rows,}\end{cases}$$

$$h'_i(X_f) = \begin{cases} 1, & \text{for rows, for which the row pattern is}\\ & \text{described by the function} \overline{g_i(X_b)},\\ 0, & \text{for all other rows ,}\end{cases} \tag{4}$$

where $i = 1, \dots, \mu(X_f|X_b)$. Using these functions, and assuming that the row multiplicity is $p$, the following equation is derived:

$$\begin{aligned}
&\mu(X_f|X_b) = p \Rightarrow f(X) = h_0(X_f)\\
&\quad + \sum_{i=1}^{p} \Big[h_i(X_f) \cdot g_i(X_b) + h'_i(X_f) \cdot \overline{g_i(X_b)}\Big].
\end{aligned} \tag{5}$$

In the case of the discussed example of a function with variables partition $X_f = \{x_0, x_1, x_2\}$, $X_b = \{x_3, x_4, x_5\}$, the functions $h$ are expressed as follows:

$$\begin{aligned}
&h_0(X_f) = x_0\overline{x}_1x_2,\\
&h_1(X_f) = \overline{x}_0\,\overline{x}_1\overline{x}_2 + \overline{x}_0x_1x_2 + x_0x_1\overline{x}_2,\\
&h'_1(X_f) = \overline{x}_0x_1\overline{x}_2,\\
&h_2(X_f) = \overline{x}_0\overline{x}_1x_2, \qquad h'_2(X_f) = 0,
\end{aligned}$$

and, finally, the function $f$ will be of the form

$$\begin{aligned}
f(X) &= x_0\overline{x}_1x_2 + (\overline{x}_0\,\overline{x}_1\overline{x}_2 + \overline{x}_0x_1x_2\\
&\quad + x_0x_1\overline{x}_2)g_1(x_3, x_4, x_5)\\
&\quad + \overline{x}_0x_1\overline{x}_2\,\overline{g_1(x_3, x_4, x_5)}\\
&\quad + \overline{x}_0\,\overline{x}_1x_2\,g_2(x_3, x_4, x_5).
\end{aligned} \tag{6}$$

Figure 5 depicts the implementation of the function under consideration into a CPLD with six product terms in one PAL-based block. As can be seen, three PAL-based logic blocks were employed here. To compare the obtained

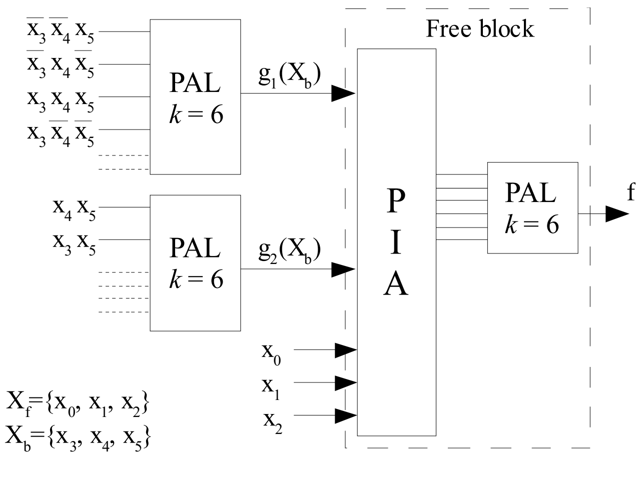


Fig. 5. Circuit after decomposition.

result with the classical one, the formula (1) is used. After two-level minimization is done with the use of the Espresso algorithm, the function under consideration (Fig. 4) can be represented as a sum of 21 products. The required number of logic blocks $\delta_f$ utilized in the classical approach, determined by the expression (1), where $k = 6$ and $\Delta_f = 21$ , is equal to 4. The classical approach requires only four PAL-based logic blocks with six product terms, i.e., one more block when compared with the decomposition-based approach.

The main idea of two-stage PAL decomposition is to search for such variable partitions which could provide

(i) a free block realisation in one PAL-based logic block, and

(ii) the smallest number of bound block outputs.

The number of bound block outputs is equal to the row multiplicity. An effective computation of the row multiplicity for a given variable partition is a major problem in two-stage PAL decomposition algorithms.

**2.1. Method for row multiplicity evaluation.** In this chapter, an algorithm for determining the row multiplicities is presented. The algorithm uses a specific colouring algorithm for the row incompatibility and complement graph. First, it is necessary to introduce some new terms. A pair of cells $(i, j)$ located in the same column of a Karnaugh map will be called **incompatible** if the values of the functions described by these cells are equal to (1,0) or (0,1). If in the set of all cell pairs located in two specific rows at least one pair of incompatible cells can be found, such rows will also be called **incompatible**. If in the set of all cell pairs located in two specific rows neither the (1,1) nor the (0,0) pairs can be found, we will say that one of the rows is a **complement** to the other. Let us now define a graph G($\mathbf{Y}$,$\mathbf{U}$), where $\mathbf{Y}$ is the set of nodes corresponding to the Karnaugh map rows, and $\mathbf{U}$ is the set of edges. Let $\mathbf{U} = \mathbf{U}_I \cup \mathbf{U}_C$, where

- $\mathbf{U}_I$ is the set of edges connecting the nodes corresponding to mutually incompatible row pairs, except for empty rows, full rows, and mutually complementing row pairs;
- $\mathbf{U}_C$ is the set of edges connecting the nodes corresponding to mutually complementing row pairs, except for empty rows and full rows.

The edges belonging to the $\mathbf{U}_I$ set will be drawn using a solid line, and the edges belonging to the $\mathbf{U}_C$ set—with a dashed line. The graph G($\mathbf{Y}$,$\mathbf{U}$), built according to the procedure described above, will be referred to as the **row incompatibility and complement graph**.

**Example 1.** Let us sketch the row incompatibility and complement graph for the function $f : B^6 \to B$ presented in Fig. 4. First, we have to locate empty rows and full rows, which will be excluded from further operations. In our case, we find two empty rows and one full row, marked in Fig. 4 with the **0** and the **1** symbol, respectively.

The analysis of subsequent row pairs leads to creating the row incompatibility and complement graph depicted in Fig. 6.

A method of the colouring of the **row incompatibility and complement graph**, presented below, allows determining the row multiplicity $\mu(\mathbf{X}_f|\mathbf{X}_b)$ of the Karnaugh map. The row multiplicity will be equal to the number of different colours, used to distinguish the nodes of the graph. The concept of graph colouring consists in assigning a minimum number of colours to graph nodes in such a way that any two nodes connected by a solid line will receive different colours. The node colouring algorithm is based on a sequential selection of nodes. A node is assigned either a **permitted** colour (denoted by a capital letter) or a **complementary** colour (denoted by a slash with capital letter). After assigning in step $i$ a permitted or complementary colour "A" to node $N$, all nodes connected to $N$ with a solid line are assigned the **forbidden** colour "a" (forbidden colours are denoted by lower case letters), and all nodes connected to $N$ with dashed lines will receive the colour complementary to "A".

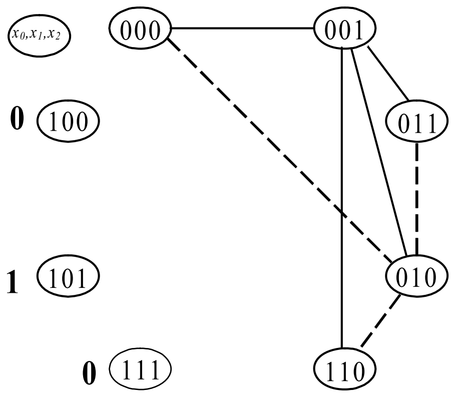


Fig. 6. Row incompatibility and complement graph.

The selection of the $i$-th node is performed according to the following rules:

- A node with the maximum number of forbidden colours is selected. It is assigned a permitted colour (one of the colours that have already been used, if possible).
- If some nodes have the same number of forbidden colours, the one to which the maximum number of edges are connected is selected.
- If some nodes have the same number of forbidden colours and the same number of edges connected, the one selected has the maximum number of complementary colours. If possible, a complementary colour is assigned to it.
- If some nodes have the same number of forbidden colours, complementary colours, and edges, the one to which the maximum number of solid line edges are connected is selected.

In the first step, node 001 is selected (Fig. 7a). The assignment of colours is performed as follows:

- node 001: **permitted** colour denoted by the $A$ letter;

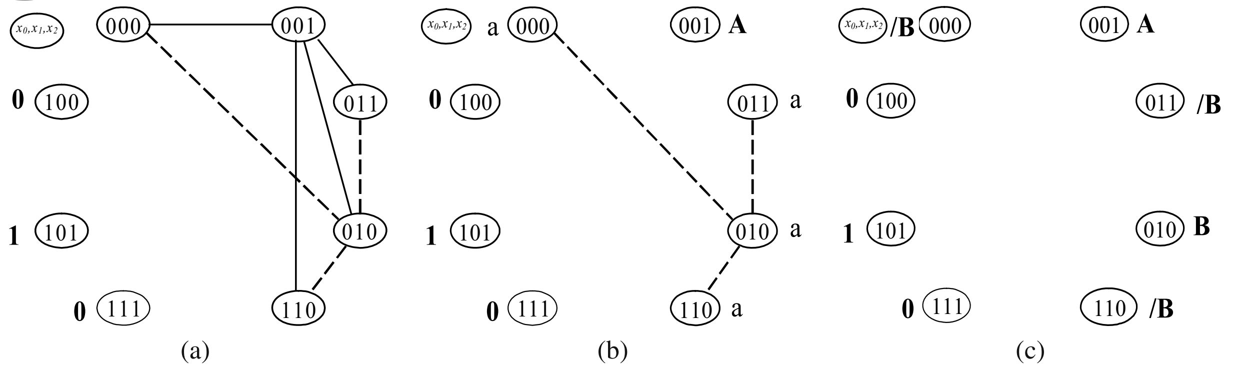


Fig. 7. Step-by-step colouring of the row incompatibility and complement graph.

- nodes 000, 011, 010, 110: **forbidden** colour denoted by the $a$ letter.

After selecting a node and assigning permitted and forbidden colours to the respective other nodes, the graph is reduced by eliminating the edges that connect the selected node with other nodes of the graph. The result is presented in Fig. 7(b). Then, a new node is selected based on the analysis of the reduced graph. The subsequent steps of node colouring are depicted in Fig. 7.

As the result of the row incompatibility and complement graph colouring procedure described above, a row multiplicity $\mu(\mathbf{X}_f|\mathbf{X}_b) = 2$ is obtained. Certainly, colour A corresponds to the row $g_2(X_b)$ in Fig. 4, and colour B to the row $\overline{g_1(X_b)}$.

A detailed description of two-stage decomposition and the proposed graph colouring algorithm can be found in (Kania, 2004; Kania *et al.*, 2005). The classical graph colouring algorithms are presented in (Chartrand and Zhang, 2008).

## 3. BDD application in decomposition

The binary decision diagram is a graph-based structure used for a memory-efficient representation of logic functions. BDDs were first proposed by Akers (1978), and popularised by Bryant (1986) and Brace *et al.* (1990). Due to their implicit power to represent Boolean functions, BDDs are considered the most efficient Boolean representation known so far. BDDs are widely used in decomposition algorithms (Lai *et al.*, 1996; Yang and Ciesielski, 2002).

A BDD is a directed acyclic graph (a tree) with each node associated with a function variable. All nodes (except for terminal ones) have two outgoing edges pointing to two children nodes, one for variable value 0 and one for 1. This binary tree contains two terminal nodes termed 0-node and 1-node. The analysis of the paths connecting to the BDD terminal nodes determines the value of the function according to the values of the variables.

Only Reduced Ordered Binary Decision Diagrams (ROBDDs) have practical meaning. In an ordered BDD, the variables in all paths have the same variable order, and they occur, at most, once on every path. Reduced ordered BDDs have a minimal number of nodes for the given variable order and are canonical forms of function representation. The reduced form is obtained from an OBDD by the reduction of the same sub-graphs and through removing all redundant nodes.

There are some ROBDDs with special attributes added to the edges for efficient memory use and faster computations (Minato, 1996). A complement is one of the most known attributes. If the edge is complemented, it means that the sub-diagram pointed by this edge must be interpreted as a negation of the formula represented by the sub-diagram.

The classical two-stage PAL decomposition employs a partition matrix as a representation of the logic function. There is a possibility to develop an algorithm using reduced ordered binary decision diagrams as an effective representation, followed by non-disjunctive decomposition, whilst the application of the negation attribute can additionally increase the algorithm's efficiency.

### 3.1. Counting the number of paths.

As far as the synthesis of digital circuits in programmable structures with PAL-based blocks is concerned, the key problem is to determine the minimal number of products in the sum of products representation. In the classical approach, the Espresso algorithm may be used for this purpose. When the ROBDD is used for logic function representation, another concept can be exploited. Each path in the diagram obtained from a root to a leaf **1** corresponds to one product. The total number of paths can vary with different variable orderings in the diagram. Changing the variable order is a way to minimize the path number. Often, the smallest number of paths is greater than the number of products after minimization, although decomposition with path counting can provide better results than the classical approach with the two-level Espresso minimization. The main advantage of the method used to determine the number of products through counting the paths is the low computa-

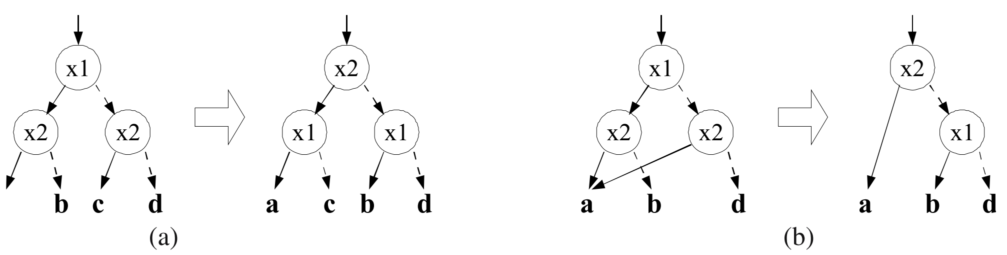


Fig. 9. Example of variable swapping in BDD ordering.

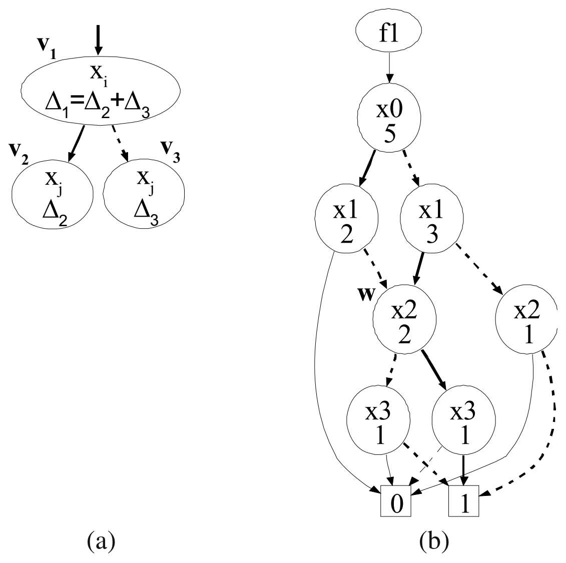


Fig. 8. Counting the number of paths.

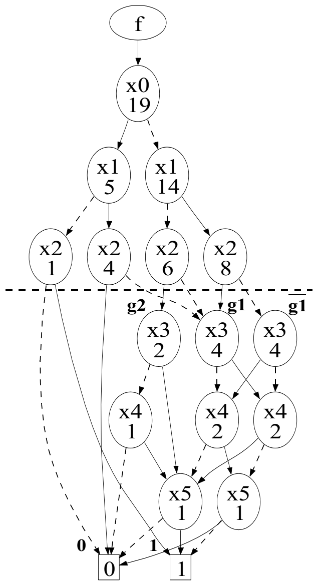
tion complexity. The number of paths can be counted by a recursive procedure. The number of paths $\Delta_1$ connecting the given node $\mathbf{v_1}$ to the leaf node **1** is equal to the sum of the number of paths connecting the children node $(high(\mathbf{v_1}), low(\mathbf{v_1}))$ to the leaf node **1** (Fig. 8a). Similarly to the standard procedure bdd_apply() (Bryant, 1986), a computed table is used to store the intermediate and final results of each algorithm iteration. A result in this context means the number of paths for a given node, which is the root of a sub-graph representing a function. Due to the use of cached intermediate results, the path counting procedure will be performed only once for each node. For instance (see Fig. 8(b)), for a node denoted by **w**, the number of paths will be computed only once, although two edges point to this node and during a depth first traversal across the diagram this node will be visited twice. The computation complexity of the procedure counting the number of paths is $O(n)$, where $n$ is the number of nodes in the diagram.

The number of paths in the diagram highly depends on the variable order. It is possible to use heuristic algorithms similar to those aiming at minimizing the number of nodes for the number of paths minimization (Ebendt *et al.*, 2005). For this purpose, a sifting algorithm (Rudell, 1993) can be used, but the optimality criterion must be changed. Each variable is moved up and down in the variable order and the position that produces the smallest OBDD size is maintained. At each position, the resulting ROBDD size is recorded and, finally, the variable is moved to the best position. The ordering change is performed by swaps of variables, which are adjacent in the variable ordering. The variable swapping affects the BDD structure of only two levels involved in the swap, whilst the whole part of the ROBDD above and below these levels remains unchanged. All modifications have a local scope and concern two levels of nodes. This local-level of the swap operation is responsible for the efficiency of the sifting algorithm. Figure 9 illustrates some portions of the ROBDDs before and after the swap operation on two levels with assigned variables $x_1$ and $x_2$. The number of nodes and paths after swapping is unchanged (see Fig. 9(a)) and changed (see Fig. 9(b)), respectively. In the second case, there is a need to recompute the total number of nodes in the diagram. Since only two levels are altered, only the number of nodes in two levels must be recounted, and the difference between the nodes number before and after swapping is added to the previous total number of nodes in the diagram. In order to compute the new number of paths in the diagram, the results of the previous calculation can also be employed. Exchanging two adjacent levels has no influence on the number of paths below these levels. The number of paths for all nodes in the upper part of the diagram must be updated. In this case, only the processing time of the lower part of the diagram is saved.

## 3.2. PAL-oriented BDD-based decomposition.

The core of PAL-oriented decomposition is to search for a partition of function variables assuring free block implementation in one PAL-based block with a constrained number of product terms. Furthermore, the partition found must provide a structure with the smallest possible number of outputs of the bound block. The partitioning of the variables in a partition matrix is equivalent to the cut in the ROBDD representing the logic function. The variables associated with the nodes above the cut line form a free set $X_f$, and below the cut line—a bound set $X_b$ (contrary to the Ashenhurst-Curtis decomposition using ROBDD representation).

Figure 10 depicts the ROBDD corresponding to the Karnaugh map of the function under consideration in Fig. 4. All nodes pointed by edges crossed by the cut line will be termed the cut nodes. As can be seen, each cut node is associated with one row pattern. The row multiplicity $\mu(X_f|X_b)$ is the number of row groups. A row group is formed by a row pattern or its complement. All nodes in an ROBDD with edge complement attributes correspond to one row group. The row multiplicity can be efficiently computed by counting the number of cut nodes in a ROBDD with edge complement attributes. Different partitions are obtained by changing the variable ordering in the ROBDD and fixing the level of the cut line diagram.



$$\mathbf{g1} \rightarrow g_1(x_3, x_4, x_5)$$
$$\mathbf{g2} \rightarrow g_2(x_3, x_4, x_5)$$
$$\begin{aligned} f = {} & x_0\bar{x}_1x_2 \\ & +\bar{x}_0\bar{x}_1\bar{x}_2\ g_1 \\ & +\bar{x}_0x_1x_2\ g_1 \\ & +x_0x_1\bar{x}_2\ g_1 \\ & +\bar{x}_0x_1\bar{x}_2\ \bar{g}_1 \\ & +\bar{x}_0\bar{x}_1x_2\ g_2 \end{aligned}$$

Fig. 10. Diagram with the number of paths of the function under consideration.

The decomposition algorithm consists of some phases. During each phase, there is established the number of free set variables which corresponds to the cut level. A variable partition is searched, which allows us to obtain a free block in one PAL-based logic block and the smallest number of the bound block outputs. If in a given phase the solution is found, the cut level is incremented.

The main idea of the decomposition algorithm is presented in Fig. 11. During each phase of the algorithm the cut level is fixed. Searching for an appropriate partition is started with the cut level equal to $\log_2(k)$, where $k$ is the number of terms in a PAL-based block. For such a cut level, a partition can always be found, so it is a good starting point. Furthermore, the cut level is incremented until the

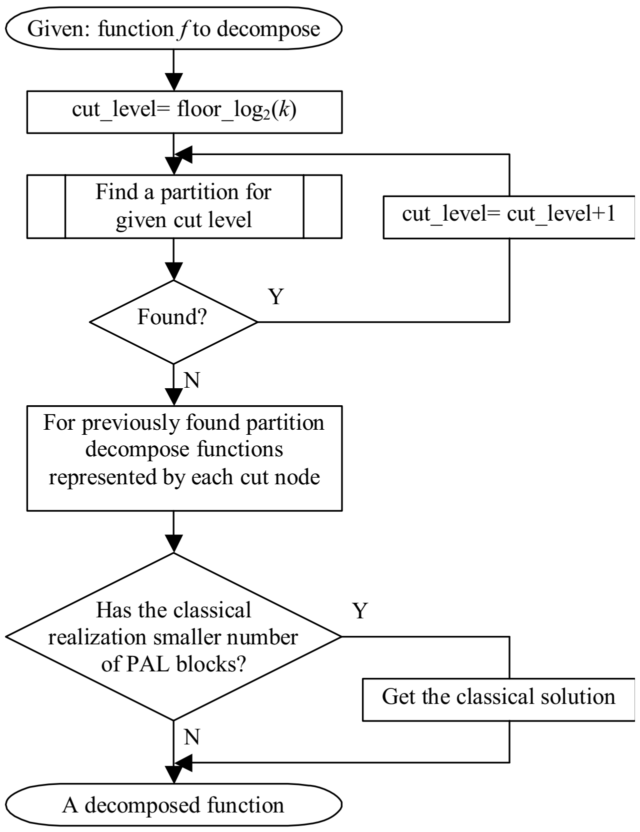

Given: function f to decompose
cut_level= floor_log2(k)
Find a partition for given cut level
cut_level= cut_level+1
Found?
Y
N
For previously found partition decompose functions represented by each cut node
Has the classical realization smaller number of PAL blocks?
Y
Get the classical solution
N
A decomposed function

Fig. 11. Simplified schema of PAL decomposition with BDD application.

partition can be found. For the last found partition a set of cut nodes is remembered and functions represented by cut nodes are further recursively decomposed. The last step of the algorithm is a comparison with the classical realisation. If the classical realisation gives a smaller number of PAL-based blocks, then the classical solution is used.

More detailed listing of the decomposition algorithm is presented in Fig. 12. The minimal number of products in the sum of products form is determined by path counting (line 4) in the ROBDD. There, the number of paths to the leaf **1** and **0** is counted. Since the relation of row complementing is symmetrical, there is a possibility to assign a function or its complement to the rows, and to obtain the solution with a smaller number of paths (products). In the case of using PAL-based logic blocks, without the possibility to program the output polarities when the decomposition procedure is initially employed, only positive polarisation should be accepted.

The algorithm contains a few improvements by just eliminating certain situations which otherwise would be further processed, e.g., if a function after minimization is described by less than $2k$ implicants (line 5) (where $k$ is the number of product terms in PAL-based logic block), then the decomposition will not reduce the number of

```
1. void decBDD( bdd f,     //decomposed function diagram
2.   int &blocks,          // # of PAL blocks
3.   int k){               //product term # in PAL block

4. int min_prod=prod_bdd(&f);//minimal # of paths (products) to leaf 0 or 1
5. if(min_prod < 2*k){
   //---------better classical realisation--------------
6.     return classical realisation;
7. }else{

   //=========decomposition============================
8.     int cut_level= floor_log2(k) + 1;
9.     //searching for best partition with fixed cut level
10.    if(find_part(&f, cut_level)){ //if found
11.        do{
12.            cut_level++;          //searching for better solutions
13.        }while(find_part(&f, cut_level));
14.        cut_level--;
15.    }else{
16.        cut_level= floor_log2(k); // log2 k +1 not found
17.        find_part(&f, cut_level);
18.    }
       //for comparison classical algorithm results
19.    int blocks_class= blocks_classical(min_prod,k);
20.    if(blocks_class <= number_of_cut_nodes+1){
       //------better classical realisation-------------
21.        return classical realisation;
22.    }else{
       //-------cut nodes decomposition------------------
23.        int tmp_blocks;           //temporary var.
24.        for(ftmp d’ż˝ cut_nodes_set){
25.            decBDD(ftmp, &tmp_blocks, k);
26.            blocks += tmp_blocks;
27.        }
28.        blocks+=1;                //free block
29.        if(blocks >= blocks_class)
30.            //---better classical realisation-------------
31.            return classical realisation;
32.    }
33. }//======end decomposition==============================
34.}
```

Fig. 12. Algorithm of PAL decomposition with BDD application.

blocks because one PAL-based logic block is needed for the free block and at least one block for the bound block, respectively. After this condition is not met, a partition is searched (lines 8–18).

A free block is ensured in one PAL-based logic block for all partitions for which $2^{|X_f|} < k,\ |X_f| \leqslant \lfloor \log_2 k \rfloor$, holds hence to save computation time the search process will start with $|X_f| = |\log_2 k| + 1$ (line 8). If the partition is found for a given cut level, the cut level is incremented (line 12). If the partition is not found for the cut level equal to $|\log_2 k| + 1$, then the partition is computed with the cut level equal to $|\log_2 k|$.

Additionally, the number of blocks in the classical approach (Formula 1) is computed in the line 19.

After a partition is found, a check will be done to see if the partition could reduce the number of PAL-based logic blocks compared to the classical approach (line 20). The minimal number of PAL-based logic blocks required to implement a circuit after partitioning is equal to $\mu(X_f|X_b)+1$, so the condition necessary to eliminate some partitions from further processing is given as

$$\mu(X_f|X_b) < \delta_f = \left\lceil \frac{\Delta_f - k}{k-1} \right\rceil + 1. \tag{7}$$

If the above condition is not true, further processing is to be continued and the functions represented by cut nodes will be decomposed (lines 24–26). At the end, the last check is made (line 29) if the decomposed function truly gives a smaller number of blocks than the classical approach.

**3.3. Algorithm refinements.** The number of paths in an ROBDD connecting the root to the leaf **1** in some cases can significantly differ than the number of product terms of two-level minimized logic function, e.g., a function with two products $f_0 = x_0x_1x_2 + x_3x_4x_5$ has four paths with variable ordering $x_0, x_1, x_2, x_3, x_4, x_5$ (Fig. 13). Using these paths, this function can be represented as a sum of four products $f_0 = x_0x_1x_2 + \overline{x_0}x_3x_4x_5 + x_0\overline{x_1}x_3x_4x_5 + x_0x_1\overline{x_2}x_3x_4x_5$. Although this representation is not optimal (as the experiments on benchmarks prove), path counting decomposition gives good results (Table 1). For further enhancement of the algorithm, the Espresso algorithm was used instead of path counting. Looking at the algorithm in Fig. 12, the only difference is in line 4, where a two-level minimization algorithm is employed as an alternative.

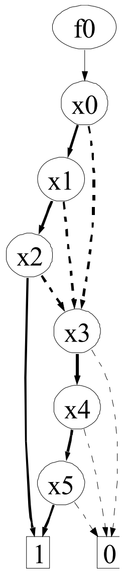


Fig. 13. Diagram with bold paths to leaf “1”.

**3.4. Non-disjunctive PAL decomposition.** In order to reduce the number of logic levels, non-disjunctive partitions can be employed (Opara, 2009; Opara and Kania, 2009). Non-disjunctive decomposition is that of the function under consideration (Fig. 14) implemented with PAL-based blocks containing three product terms. The first stage of non-disjunctive decomposition is to find a good disjunctive partition. For a given variable order, only $x_0$ can be included into the free set. In this case, a free block described as $f = x_0 \cdot g_2(x_1, x_2, x_3, x_4) + \overline{x_0} \cdot g_1(x_1, x_2, x_3, x_4)$ is implemented by two product terms. Function $g_2$ describes a diagram rooted by node $\mathbf{v_2}$, and $g_1$ by $\mathbf{v_1}$. Due to the inclusion of one more variable ($x_1$) to the disjunctive free set, four product terms are needed, so the limit of three terms in a PAL block is exceeded. Function $g_1$ is created in one PAL block and $g_2$ in two blocks, respectively. Finally, using disjunctive decomposition, a circuit can be implemented with four blocks situated in three levels.

Through the introduction of non-disjunctive decomposition, the variable $x_1$ is included into the free and bound set. The free block is described by the formula $f = x_0 \cdot x_1 \cdot g_0 + x_0 \cdot \overline{x_1} \cdot \overline{g_0} + \overline{x_0} \cdot g_1$ and utilizes three product terms. The whole circuit is built of three PAL-based logic blocks in two levels (Fig. 15).

The algorithm presented in Fig. 12 is modified, so after a proper disjunctive partition is found (lines 23–34) a procedure is employed to try to add one child of a cut node to the cut node set. In the example considered, $\mathbf{v_0}$ is chosen as a child of $\mathbf{v_1}$. The node is accepted if the resulting implementation of the free block fits one PAL-based logic block.

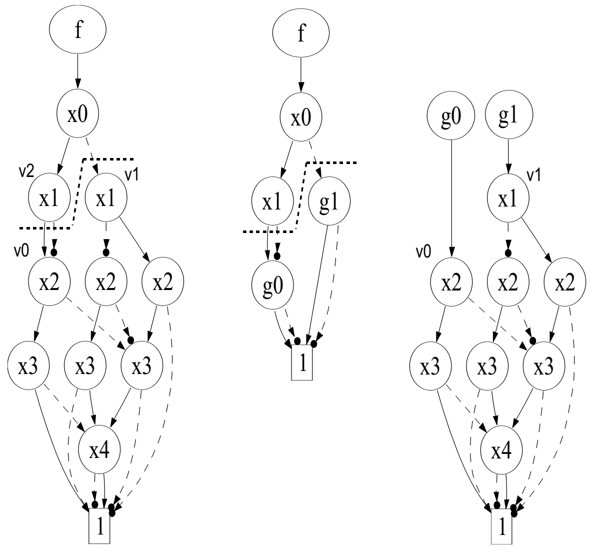


Fig. 14. Diagram cut corresponding to non-disjoint decomposition.

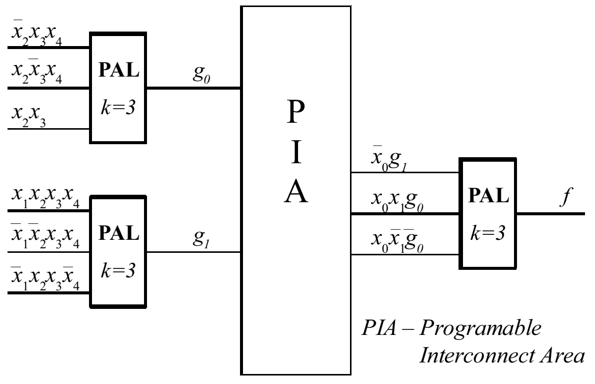


Fig. 15. Circuit structure after non-disjunctive decomposition.

## 4. Experimental results

The developed BDD-based synthesis methods were compared with

(i) the classical method,

(ii) a two-stage decomposition, and

(iii) a synthesis implemented in firmware tools (Quartus).

In order to compare these methods, a synthesis of benchmarks was carried out for PAL-based logic blocks containing $k$ number of terms. All experiments were performed on a PC with a Pentium Centrino 1,6 GHz processor and 1 GB RAM, under the Windows XP operating system. To carry out the experiments, a dekBDD prototype tool was developed, about which some additional information can be found at: `db.zmitac.aei.polsl.pl/AO/dekBDD.html`.

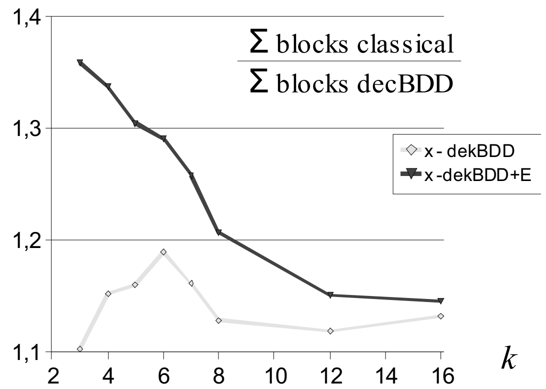


Fig. 16. Comparison of two algorithms with the classical method with respect to the number of logic blocks (see the text).

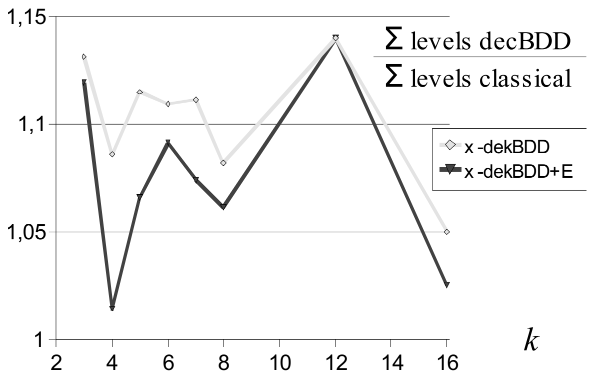


Fig. 17. Comparison of two algorithms with the classical method with respect to the number of logic levels (see the text).

### 4.1. Comparison with the classical method.

A method of implementing a function in PAL-based structures incorporating the BDD presented in this paper (dekBDD) was compared with the classical approach with respect to the number of logic blocks used and the number of logic levels. The comparison was made for an algorithm in two versions: simple, denoted by dekBDD, and enhanced, denoted by dekBDD+E in Table 1. For multi-output benchmarks, this algorithm was applied separately to the outputs. The left part of the table shows the results of the synthesis performed on the benchmarks using the classical approach. The column marked with "Esp" lists the number of function products after the Espresso minimization, "Bdd" lists the number of paths in the ROBDD representing the function, the letter "B" list the numbers of $k$-product PAL-based blocks, and the columns marked with the letter "L" list the numbers of logic levels.

The second part of the table, denoted with the heading "dekBDD", contains the results obtained using the new method. In the set of about 2600 cases compared, the proposed dekBDD algorithm allowed 168 solutions to be found, whilst the dekBDD+E algorithm allowed 263 solutions to be found, which required a smaller number of logic blocks than in the classical method. For some benchmarks, the reduction of the logic block count was significant, e.g., for rd84 $f_1$, rd73 $f_1$, cordic $f_1$, misex3 $f_2$, $f_7$, 5xp1 $f_2$. Significant differences can be noticed not only for small values of $k$. Unfortunately, the number of logic levels does not follow the reduction of the number of logic blocks. Among the examined benchmarks, only a few percent of the solutions demanded a smaller number of logic levels.

The results of the experiments are presented in a synthetic way in Figs. 16 and 17. The values represented on the axis of ordinates in Fig. 16 were calculated from the rational formula shown in the graph. $\Sigma$blocks classical and $\Sigma$blocks dekBDD denote the relevant total sums of block counts obtained using the corresponding synthesis methods and are presented in Table 1. The values represented in Fig. 17 were calculated in a similar manner. The analysis of the benchmarks allows us to state that, in most cases, the reduction of logic block counts by using this new algorithm is obtained at the expense of a certain expansion of logic levels. The proposed method is particularly efficient if $k = 4$, 8, and 16. A significant reduction in block counts was observed, while preserving a comparable number of logic levels.

### 4.2. Comparison with two-stage decomposition.

In the development of BDD-based decomposition algorithms, two-stage decomposition was a certain reference. This method was presented in Section 2. Decomposition based on the classical Ashenhurst-Curtis model is very effective. Unfortunately, the computation complexity precludes the synthesis of large designs. Two-stage decomposition implemented in a PALDec system (Kania, 2004) allows the synthesis of functions with at most 16 arguments. A logic synthesis based on a BDD (a simple one, denoted by dekBDD, and the enhanced one, denoted by dekBDD+E) was compared with two-stage decomposition implemented in a PALDec system with respect to the number of logic blocks used and the number of logic levels. The results are presented in Table 2. The rows

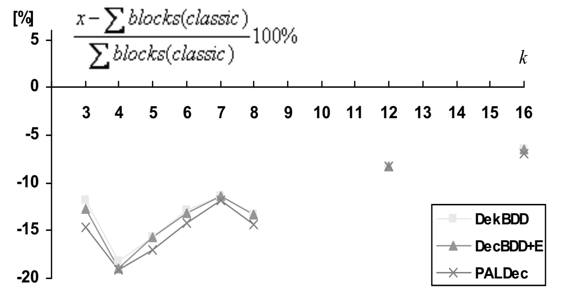


Fig. 18. Comparison of decomposition algorithms with respect to the number of logic blocks (see the text).

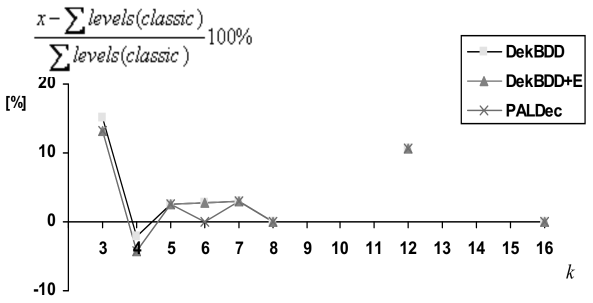


Fig. 19. Comparison of decomposition algorithms with respect to the number of logic levels (see the text).

show the results of synthesis performed on the benchmarks using the classical approach (rows marked “Classic”), logic synthesis based on the BDD (rows marked “dekBDD” and “dekBDD+E”) and logic synthesis based on two-stage decomposition (rows marked “PALDec”). The columns marked with the letter “B” list the numbers of $k$-product PAL-based blocks used, and the columns marked with the letter “L”—the numbers of logic levels.

The relevant total sums of block and level counts obtained using the corresponding synthesis methods are presented in the four lowest rows of Table 2.

When comparing a set of 128 cases, two-stage decomposition (PALDec) gave 20 solutions (15%) requiring a smaller number of logic blocks than BDD-based decomposition methods. For certain benchmarks, the reduction of logic block count was significant, e.g., for *f51m, z5xp*. Crucial differences can be noticed only for $k = 3$. In the majority of cases, the numbers of logic blocks and levels obtained for both methods were identical.

The results of the comparison of all decomposition methods are presented in a synthetic form in Figs. 18 and 19. The values represented on the axis of ordinates in Fig. 18 and Fig. 19 were calculated from the formula shown on the graph. $\Sigma blocks$ and $\Sigma levels$ denote the relevant total sums of block counts obtained with the use of the corresponding synthesis methods.

The analysis of the benchmarks allows us to state that, in most cases, the reduction of logic block counts by using the new algorithm is obtained at the expense of a certain expansion of logic levels. All decomposition methods are particularly efficient if $k = 4$ or $k = 8$. In this case, a significant reduction of block counts, while preserving comparable (sometimes the lowest) number of logic levels, was observed.

These experiments show the following:

- The decomposition method is better, with respect to the number of logic blocks, than the classical approach.
- The decomposition method can be useful in cases for which the reduction of the chip area is of the utmost concern, without significantly degrading the chip dynamic properties.
- BDD based decomposition algorithms (DekBDD, DekBDD+E) have significantly lower computation complexity than two-stage decomposition based algorithms.
- Two-stage decomposition sometimes produces better solutions than BDD-based techniques.
- If the reduction of the number of logic levels is an important factor in the synthesis, the proposed decomposition algorithm is particularly effective for structures consisting of PAL-based blocks containing $2^i$ (a power of 2) product terms.

**4.3. Way to describe circuits to use decomposition results for commercial applications.** The main problem in porting a proposed method 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) use low level netlists for this purpose. This approach is secure because there is a 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 requires thus equipping the synthesis software with procedures or plugins responsible for converting formats, and preparing data specific for the implementation tools. This is acceptable for commercial companies but difficult for academic experiments. It was, therefore, desirable to find alternative formats for the data exchange, possibly more universal, and to use a higher level of abstraction.

Here using a Hardware Description Language (HDL) seems to be the most obvious 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 presently to be the design specification format most preferred for design entry. 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. The benchmarks were implemented in PAL-based CPLDs.

It was found that, if behavioural description was used as the entry format, the quality of the solutions was not good. A high abstraction level in behavioural modelling gives much freedom to the software. Logical structures can be easily "spoiled" by vendor implementation programs. During the experiments it appeared that it is possible to propose as the intermediate format a style of Verilog description lying at a lower level of abstraction than behavioural modelling, but still portable between software tools and comprehendible to a human.

To this end, a way to describe the circuit under design was developed using a set of equations. The advantage of this solution is that the decomposed circuit retains its structure. The proposed circuit description ensures the transferability of results to different hardware platforms. The designed circuit described by the sum of products (Fig. 20(a)) is then decomposed using the prototype tool, to obtain the description in the form of a set of equations in the Verilog language (Fig. 20(b)). Adding certain attribute signals (* KEEP *) prevents a specific given signal from being reduced by the firmware synthesis tool. In consequence, the decomposed circuit will retain its specific structure. Apart from the transferability, such a description has another advantage—it does not limit the possibility of using specific resources of programmable structures, such as, e.g., shared expanders. Thus, further improvement of the obtained results is possible and experiments confirmed the effectiveness of the proposed approach. In order to verify the practical usefulness of the proposed description and decomposition methods for standard benchmarks, the Quartus II v8.0 software from Altera and the MAX 7000B, EPM 7512 BFC256-5 programmable logic were used. Each benchmark was synthesised using the Quartus software. Moreover, a description of the circuit was produced for comparison purposes, using dekBDD, and then the synthesis was continued using the firmare system.

```
5xp1.pla
module benchm (
input x0,x1,x2,x3,x4,x5,x6,
output
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9
);
⋮
assign f2 =
(x0&~x4&x5&~x6)
|(~x0&~x4&~x5&x6)
|(~x1&~x2&~x4&~x5&x6)
|(x0&x1&x2&~x4&~x6)
|(x0&x1&x3&~x4&~x6)
|(~x0&x4&~x5&~x6)
|(~x1&x4&~x5&~x6)
|(~x2&~x3&x4&~x5&~x6)
|(x1&x2&x3&~x4&x5&~x6)
|(~x0&~x1&~x3&~x4&x6)
|(~x0&~x1&~x2&~x4&x6)
|(x0&x4&x5&x6)
|(x1&x4&x5&x6)
|(~x0&~x1&x2&x3&x4&x5)
|(~x0&~x3&x4&~x6)
|(~x0&~x2&x4&~x6)
|(x0&x1&x4&x6)
|(x0&x2&x4&x6);
⋮
endmodule
```

(a)

```
5xp1.pla
module benchm (
input x0,x1,x2,x3,x4,x5,x6,
output
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9
);
⋮
(* KEEP *) wire g20 = ~(
(x1&~x6&x2&x3)
|(~x1&x6&~x0&~x3)
|(~x1&x6&~x0&~x2)
|(~x6&x0) );

(* KEEP *) wire g21 =
~( (x1&~x6&x0&x3)
|(x1&~x6&x0&x2)
|(~x1&x6&~x2)
|(x6&~x0) );

assign f2 =
(~x4&x5&~g20)
|(x4&x5&g20)
|(~x4&~x5&~g21)
|(x4&~x5&g21);

⋮
endmodule
```

(b)

Fig. 20. Description of the function `f2 5xp1` in the Verilog language: classical representation (a), representation created with the prototype tool (allowing the use of the decomposition results (b).

The results of the experiments are presented in Table 3. The first column header contains the name of the benchmark, the next two columns contain the number of the blocks of type PAL ($k = 5$) obtained using the classical method and the dekBDD + E method. The next two groups of column headers contain "MAX 7000B Quartus II area opt." and "MAX 7000B dekBDD + E + Quartus II", and they are related to the results of the synthesis of the circuits developed in the MAX 7000B programmable structure using two methods:

1. "MAX 7000B Quartus II area opt."—synthesis employing Quartus, focused on area minimization;
2. "MAX 7000B dekBDD + E + Quartus II"—decomposition using the dekBDD + E method ending with the description in the Verilog language, followed by post-synthesis using Quartus.

Here, "MC" is the number of the macrocells, "Exp" is the number of the shared expanders used, and "tp" is the propagation time through the longest path. The penultimate column contains the standardized number of macrocells

Table 1. Detailed comparison between the classical method referred to decompositions that employ the BDD.

| | | | Classic | | | | | | | | | | | | | | | | DekBDD | | | | | | | | | | | | | | | | Gain | DekBDD+E | | | | | | | | | | | | | | | | Gain |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | k=3 | | k=4 | | k=5 | | k=6 | | k=7 | | k=8 | | k=12 | | k=16 | | k=3 | | k=4 | | k=5 | | k=6 | | k=7 | | k=8 | | k=12 | | k=16 | | | k=3 | | k=4 | | k=5 | | k=6 | | k=7 | | k=8 | | k=12 | | k=16 | | |
| | Esp | Bdd | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L | |
| f2: | 18 | 19 | 9 | 3 | 6 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 6 | $\underline{4}$ | 5 | 3 | 3 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 5 | 3 | 3 | 2 | 3 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f3: | 14 | 15 | 7 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 4 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | | 4 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | |
| f4: | 10 | 10 | 5 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | | 3 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | |
| 5xp1.pla | | | 35 | 3 | 26 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 27 | $\underline{4}$ | 23 | 3 | 19 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 14 | 26 | 3 | 21 | 2 | 19 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 17 |
| 9sym.pl | 86 | 148 | 43 | 5 | 29 | 4 | 22 | 3 | 17 | 3 | 15 | 3 | 13 | 3 | 8 | 2 | 6 | 2 | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | $\underline{3}$ | 5 | 2 | 34 | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 | 34 |
| f0: | 21 | 22 | 10 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 9 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 9 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f2: | 42 | 54 | 21 | 4 | 14 | 3 | 11 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 4 | 2 | 3 | 2 | 21 | 4 | 13 | 3 | 11 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 4 | 2 | 3 | 2 | | 21 | 4 | 13 | 3 | 11 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 4 | 2 | 3 | 2 | |
| f3: | 34 | 43 | 17 | 4 | 11 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | 17 | 4 | 9 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | | 15 | 4 | 9 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | |
| f4: | 20 | 21 | 10 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 6 | 3 | 5 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 6 | 3 | 5 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| clip.pla | | | 73 | 4 | 49 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 | 68 | 4 | 44 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 | 10 | 66 | 4 | 44 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 | 12 |
| f0: | 23 | 23 | 11 | 3 | 8 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 4 | 2 | 2 | 2 | 2 | 2 | 11 | 3 | 5 | 3 | 4 | $\underline{3}$ | 3 | 2 | 3 | 2 | 4 | 2 | 2 | 2 | 2 | 2 | | 11 | 3 | 5 | 3 | 4 | 3 | 3 | 2 | 3 | 2 | 4 | 2 | 2 | 2 | 2 | 2 | |
| f1: | 18 | 19 | 9 | 3 | 6 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 9 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 9 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f2: | 14 | 15 | 7 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 7 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | | 7 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | |
| f51m.pla | | | 37 | 3 | 27 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 11 | 2 | 10 | 2 | 37 | 3 | 19 | 3 | 17 | $\underline{3}$ | 15 | 2 | 15 | 2 | 15 | 2 | 11 | 2 | 10 | 2 | 17 | 37 | 3 | 19 | 3 | 17 | 3 | 15 | 2 | 15 | 2 | 15 | 2 | 11 | 2 | 10 | 2 | 17 |
| f1: | 16 | 16 | 8 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 1 | 1 | 4 | $\underline{4}$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | |
| f2: | 10 | 14 | 5 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 5 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | | 5 | 3 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | |
| rd53.pla | | | 15 | 3 | 10 | 2 | 8 | 2 | 6 | 2 | 6 | 2 | 6 | 2 | 4 | 2 | 3 | 1 | 11 | $\underline{4}$ | 7 | 2 | 6 | 2 | 5 | 2 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 1 | 12 | 11 | 4 | 7 | 2 | 6 | 2 | 5 | 2 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 1 | 12 |
| f0: | 42 | 48 | 21 | 4 | 14 | 3 | 11 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 4 | 2 | 3 | 2 | 21 | 4 | 7 | 3 | 7 | 3 | 6 | 3 | 5 | $\underline{3}$ | 3 | 2 | 3 | 2 | 3 | 2 | | 21 | 4 | 7 | 3 | 7 | 3 | 5 | 3 | 5 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | |
| f1: | 64 | 64 | 32 | 4 | 21 | 3 | 16 | 3 | 13 | 3 | 11 | 3 | 9 | 2 | 6 | 2 | 5 | 2 | 6 | $\underline{6}$ | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | | 6 | 6 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | |
| f2: | 35 | 35 | 17 | 4 | 12 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 15 | 4 | 8 | 3 | 6 | 3 | 5 | $\underline{3}$ | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | | 15 | 4 | 8 | 3 | 6 | 3 | 5 | 3 | 5 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | |
| rd73.pla | | | 70 | 4 | 47 | 3 | 36 | 3 | 29 | 3 | 24 | 3 | 20 | 2 | 14 | 2 | 11 | 2 | 42 | $\underline{6}$ | 18 | 3 | 16 | 3 | 14 | 3 | 13 | 3 | 10 | 2 | 9 | 2 | 8 | 2 | 121 | 42 | 6 | 18 | 3 | 16 | 3 | 13 | 3 | 13 | 3 | 10 | 2 | 9 | 2 | 8 | 2 | 122 |
| f0: | 84 | 92 | 42 | 5 | 28 | 4 | 21 | 3 | 17 | 3 | 14 | 3 | 12 | 3 | 8 | 2 | 6 | 2 | 42 | 5 | 11 | 4 | 7 | 3 | 7 | 3 | 7 | 3 | 5 | 3 | 3 | 2 | 3 | 2 | | 40 | 5 | 11 | 4 | 7 | 3 | 7 | 3 | 7 | 3 | 5 | 3 | 3 | 2 | 3 | 2 | |
| f1: | 128 | 128 | 64 | 5 | 43 | 4 | 32 | 4 | 26 | 3 | 22 | 3 | 19 | 3 | 12 | 2 | 9 | 2 | 7 | $\underline{7}$ | 4 | 4 | 4 | 4 | 4 | $\underline{4}$ | 4 | $\underline{4}$ | 3 | 3 | 3 | $\underline{3}$ | 2 | 2 | | 7 | 7 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 3 | 2 | 2 | |
| f3: | 70 | 73 | 35 | 4 | 23 | 4 | 18 | 3 | 14 | 3 | 12 | 3 | 10 | 3 | 7 | 2 | 5 | 2 | 27 | $\underline{5}$ | 11 | 3 | 11 | 3 | 9 | 3 | 8 | 3 | 6 | 3 | 5 | 2 | 5 | 2 | | 27 | 5 | 11 | 3 | 11 | 3 | 9 | 3 | 8 | 3 | 6 | 3 | 5 | 2 | 5 | 2 | |
| rd84.pla | | | 142 | 5 | 95 | 4 | 72 | 4 | 58 | 3 | 49 | 3 | 42 | 3 | 28 | 2 | 21 | 2 | 77 | $\underline{7}$ | 27 | 4 | 23 | 4 | 21 | $\underline{4}$ | 20 | $\underline{4}$ | 15 | 3 | 12 | $\underline{3}$ | 11 | 2 | 301 | 75 | 7 | 27 | 4 | 23 | 4 | 21 | 4 | 20 | 4 | 15 | 3 | 12 | 3 | 11 | 2 | 303 |
| f2: | 22 | 31 | 11 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 8 | $\underline{4}$ | 6 | 3 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 6 | 3 | 4 | 3 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f3: | 21 | 33 | 10 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 6 | 3 | 7 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 5 | 3 | 6 | 3 | 5 | 2 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| sao2.pla | | | 36 | 3 | 24 | 3 | 19 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 | 29 | $\underline{4}$ | 23 | 3 | 18 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 | 9 | 26 | 3 | 20 | 3 | 18 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 | 15 |
| xor5.pla | 16 | 16 | 8 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 1 | 1 | 4 | $\underline{4}$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 12 | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 12 |
| f3: | 18 | 19 | 9 | 3 | 6 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 9 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 9 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f4: | 14 | 15 | 7 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 7 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | | 7 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | |
| z5xp1.p | | | 35 | 3 | 26 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 35 | 3 | 21 | 2 | 19 | 2 | 17 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 9 | 35 | 3 | 21 | 2 | 19 | 2 | 17 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 | 9 |
| z9sym.p | 86 | 148 | 43 | 5 | 29 | 4 | 22 | 3 | 17 | 3 | 15 | 3 | 13 | 3 | 8 | 2 | 6 | 2 | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | $\underline{3}$ | 5 | 2 | 34 | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 | 34 |
| f3: | 67 | 83 | 33 | 4 | 22 | 4 | 17 | 3 | 14 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | 33 | 4 | 22 | 4 | 17 | 3 | 14 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | | 32 | 4 | 22 | 4 | 17 | 3 | 14 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | |
| f4: | 65 | 82 | 32 | 4 | 22 | 4 | 16 | 3 | 13 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | 32 | 4 | 22 | 4 | 16 | 3 | 13 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | | 29 | 4 | 21 | 4 | 16 | 3 | 13 | 3 | 11 | 3 | 10 | 3 | 6 | 2 | 5 | 2 | |
| f5: | 73 | 89 | 36 | 4 | 24 | 4 | 18 | 3 | 15 | 3 | 12 | 3 | 11 | 3 | 7 | 2 | 5 | 2 | 36 | 4 | 24 | 4 | 18 | 3 | 15 | 3 | 12 | 3 | 11 | 3 | 7 | 2 | 5 | 2 | | 36 | 4 | 24 | 4 | 18 | 3 | 15 | 3 | 12 | 3 | 11 | 3 | 7 | 2 | 5 | 2 | |
| f6: | 77 | 92 | 38 | 4 | 26 | 4 | 19 | 3 | 16 | 3 | 13 | 3 | 11 | 3 | 7 | 2 | 6 | 2 | 38 | 4 | 26 | 4 | 19 | 3 | 16 | 3 | 13 | 3 | 11 | 3 | 7 | 2 | 6 | 2 | | 38 | 4 | 26 | 4 | 19 | 3 | 16 | 3 | 13 | 3 | 11 | 3 | 7 | 2 | 6 | 2 | |
| f7: | 85 | 102 | 42 | 5 | 28 | 4 | 21 | 3 | 17 | 3 | 14 | 3 | 12 | 3 | 8 | 2 | 6 | 2 | 42 | 5 | 28 | 4 | 21 | 3 | 17 | 3 | 14 | 3 | 12 | 3 | 8 | 2 | 6 | 2 | | 41 | 5 | 28 | 4 | 21 | 3 | 17 | 3 | 14 | 3 | 12 | 3 | 8 | 2 | 6 | 2 | |
| apex3.p | | | 312 | 5 | 222 | 4 | 175 | 3 | 151 | 3 | 128 | 3 | 119 | 3 | 89 | 2 | 79 | 2 | 312 | 5 | 222 | 4 | 175 | 3 | 151 | 3 | 128 | 3 | 119 | 3 | 89 | 2 | 79 | 2 | 0 | 307 | 5 | 221 | 4 | 175 | 3 | 151 | 3 | 128 | 3 | 119 | 3 | 89 | 2 | 79 | 2 | 6 |
| f0: | 278 | 2648 | 139 | 6 | 93 | 5 | 70 | 4 | 56 | 4 | 47 | 3 | 40 | 3 | 26 | 3 | 19 | 3 | 139 | 6 | 93 | 5 | 70 | 4 | 56 | 4 | 47 | 3 | 40 | 3 | 26 | 3 | 19 | 3 | | 139 | 6 | 93 | 5 | 70 | 4 | 56 | 4 | 47 | 3 | 40 | 3 | 21 | 3 | 19 | 3 | |
| f1: | 264 | 1227 | 132 | 6 | 88 | 5 | 66 | 4 | 53 | 4 | 44 | 3 | 38 | 3 | 24 | 3 | 18 | 3 | 132 | 6 | 88 | 5 | 66 | 4 | 53 | 4 | 44 | 3 | 38 | 3 | 24 | 3 | 18 | 3 | | 119 | 6 | 77 | 5 | 66 | 4 | 53 | 4 | 30 | $\underline{4}$ | 25 | 3 | 24 | 3 | 18 | 3 | |
| f2: | 523 | 1763 | 261 | 6 | 174 | 5 | 131 | 4 | 105 | 4 | 87 | 4 | 75 | 4 | 48 | 3 | 35 | 3 | 261 | 6 | 174 | 5 | 131 | 4 | 105 | 4 | 87 | 4 | 75 | 4 | 48 | 3 | 35 | 3 | | 131 | 6 | 74 | 5 | 57 | 4 | 46 | 4 | 38 | 4 | 23 | 4 | 22 | 3 | 17 | 3 | |
| apex2.p | | | 532 | 6 | 355 | 5 | 267 | 4 | 214 | 4 | 178 | 4 | 153 | 4 | 98 | 3 | 72 | 3 | 532 | 6 | 355 | 5 | 267 | 4 | 214 | 4 | 178 | 4 | 153 | 4 | 98 | 3 | 72 | 3 | 0 | 389 | 6 | 244 | 5 | 193 | 4 | 155 | 4 | 115 | 4 | 88 | 4 | 67 | 3 | 54 | 3 | 564 |
| f6: | 36 | 46 | 18 | 4 | 12 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 18 | 4 | 12 | 3 | 8 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | | 18 | 4 | 12 | 3 | 8 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | |
| f7: | 199 | 297 | 99 | 5 | 66 | 4 | 50 | 4 | 40 | 3 | 33 | 3 | 29 | 3 | 18 | 3 | 14 | 2 | 99 | 5 | 66 | 4 | 50 | 4 | 40 | 3 | 33 | 3 | 29 | 3 | 18 | 3 | 14 | 2 | | 55 | 5 | 46 | 4 | 34 | 4 | 28 | $\underline{4}$ | 24 | 3 | 28 | 3 | 18 | 3 | 14 | 2 | |
| alu4.pla | | | 323 | 5 | 216 | 4 | 163 | 4 | 131 | 3 | 109 | 3 | 94 | 3 | 62 | 3 | 46 | 2 | 323 | 5 | 216 | 4 | 162 | 4 | 131 | 3 | 109 | 3 | 94 | 3 | 62 | 3 | 46 | 2 | 1 | 279 | 5 | 196 | 4 | 146 | 4 | 119 | $\underline{4}$ | 100 | 3 | 93 | 3 | 62 | 3 | 46 | 2 | 103 |
| f0: | 143 | 6809 | 71 | 5 | 48 | 4 | 36 | 4 | 29 | 3 | 24 | 3 | 21 | 3 | 13 | 2 | 10 | 2 | 71 | 5 | 48 | 4 | 36 | 4 | 29 | 3 | 24 | 3 | 21 | 3 | 10 | $\underline{5}$ | 8 | $\underline{4}$ | | 71 | 5 | 48 | 4 | 33 | $\underline{5}$ | 29 | 3 | 17 | 3 | 20 | $\underline{4}$ | 6 | 3 | 4 | 3 | |
| f1: | 771 | 3563 | 385 | 7 | 257 | 5 | 193 | 5 | 154 | 4 | 129 | 4 | 110 | 4 | 70 | 3 | 52 | 3 | 385 | 7 | 198 | $\underline{8}$ | 119 | $\underline{8}$ | 34 | $\underline{7}$ | 30 | $\underline{7}$ | 48 | $\underline{6}$ | 32 | $\underline{5}$ | 14 | $\underline{5}$ | | 132 | 7 | 94 | 5 | 69 | 5 | 22 | 6 | 22 | 6 | 32 | 5 | 20 | 4 | 11 | 4 | |
| cordic.p | | | 456 | 7 | 305 | 5 | 229 | 5 | 183 | 4 | 153 | 4 | 131 | 4 | 83 | 3 | 62 | 3 | 456 | 7 | 246 | $\underline{8}$ | 155 | $\underline{8}$ | 63 | $\underline{7}$ | 54 | $\underline{7}$ | 69 | $\underline{6}$ | 42 | $\underline{5}$ | 22 | $\underline{5}$ | 495 | 203 | 7 | 142 | 5 | 102 | 5 | 51 | 6 | 39 | 6 | 52 | 5 | 26 | 4 | 15 | 4 | 972 |
| f1: | 8 | 8 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | |
| f64: | 27 | 30 | 13 | 3 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 3 | 9 | 3 | 7 | 3 | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | $\underline{5}$ | 8 | $\underline{4}$ | 6 | 3 | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f65: | 27 | 30 | 13 | 3 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 3 | 9 | 3 | 7 | 3 | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | $\underline{5}$ | 8 | $\underline{4}$ | 6 | 3 | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f68: | 27 | 30 | 13 | 3 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 3 | 9 | 3 | 7 | 3 | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | $\underline{5}$ | 8 | $\underline{4}$ | 6 | 3 | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f69: | 27 | 30 | 13 | 3 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 3 | 9 | 3 | 7 | 3 | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | $\underline{5}$ | 8 | $\underline{4}$ | 6 | 3 | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f70: | 27 | 30 | 13 | 3 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 3 | 9 | 3 | 7 | 3 | 5 | $\underline{3}$ | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | $\underline{5}$ | 8 | $\underline{4}$ | 6 | 3 | 5 | 3 | 4 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f72: | 23 | 23 | 11 | 3 | 8 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 4 | 2 | 2 | 2 | 2 | 2 | 8 | 3 | 4 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 8 | 3 | 4 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | |
| f76: | 28 | 29 | 14 | 4 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | 13 | 4 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | | 11 | 4 | 9 | 3 | 7 | 3 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 2 | 2 | |
| f80: | 32 | 39 | 16 | 4 | 11 | 3 | 8 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | 12 | $\underline{6}$ | 11 | 3 | 8 | 3 | 6 | $\underline{4}$ | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | | 12 | 6 | 11 | 3 | 8 | 3 | 6 | 4 | 6 | 2 | 5 | 2 | 3 | 2 | 3 | 2 | |
| f84: | 36 | 37 | 18 | 4 | 12 | 3 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | 12 | $\underline{6}$ | 9 | $\underline{5}$ | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | | 12 | 6 | 9 | 5 | 9 | 3 | 7 | 2 | 6 | 2 | 5 | 2 | 4 | 2 | 3 | 2 | |
| apex5.p | | | 578 | 4 | 391 | 3 | 308 | 3 | 253 | 2 | 230 | 2 | 215 | 2 | 138 | 2 | 110 | 2 | 562 | $\underline{6}$ | 383 | $\underline{5}$ | 305 | 3 | 245 | $\underline{4}$ | 229 | 2 | 214 | 2 | 138 | 2 | 110 | 2 | 37 | 550 | 6 | 378 | 5 | 300 | 3 | 245 | 4 | 224 | 2 | 214 | 2 | 138 | 2 | 110 | 2 | 64 |
| t481.pla | 481 | 841 | 240 | 6 | 160 | 5 | 120 | 4 | 96 | 4 | 80 | 4 | 69 | 3 | 44 | 3 | 32 | 3 | 45 | $\underline{8}$ | 41 | $\underline{7}$ | 32 | $\underline{7}$ | 49 | $\underline{5}$ | 47 | $\underline{5}$ | 46 | $\underline{5}$ | 21 | $\underline{4}$ | 14 | 3 | 546 | 43 | 8 | 39 | 7 | 31 | 7 | 47 | 5 | 45 | 5 | 45 | 5 | 21 | 4 | 14 | 3 | 556 |
| f0: | 87 | 99 | 43 | 5 | 29 | 4 | 22 | 3 | 18 | 3 | 15 | 3 | 13 | 3 | 8 | 2 | 6 | 2 | 43 | 5 | 27 | 4 | 22 | 3 | 15 | $\underline{4}$ | 15 | 3 | 13 | 3 | 8 | 2 | 5 | 2 | | 15 | 4 | 14 | 3 | 13 | 3 | 11 | 3 | 10 | 3 | 10 | 3 | 8 | 2 | 5 | 2 | |
| f1: | 102 | 120 | 51 | 5 | 34 | 4 | 26 | 3 | 21 | 3 | 17 | 3 | 15 | 3 | 10 | 2 | 7 | 2 | 51 | 5 | 28 | $\underline{5}$ | 26 | 3 | 21 | 3 | 17 | 3 | 15 | 3 | 10 | 2 | 6 | 2 | | 25 | 4 | 16 | 3 | 15 | 3 | 17 | 3 | 14 | 3 | 12 | 3 | 8 | 2 | 6 | 2 | |
| f2: | 120 | 169 | 60 | 5 | 40 | 4 | 30 | 3 | 24 | 3 | 20 | 3 | 17 | 3 | 11 | 2 | 8 | 2 | 60 | 5 | 39 | $\underline{5}$ | 30 | 3 | 23 | $\underline{4}$ | 20 | 3 | 17 | 3 | 11 | 2 | 8 | 2 | | 18 | 4 | 17 | 3 | 16 | 3 | 12 | 3 | 12 | 3 | 9 | 3 | 11 | 2 | 8 | 2 | |
| f3: | 132 | 193 | 66 | 5 | 44 | 4 | 33 | 4 | 27 | 3 | 22 | 3 | 19 | 3 | 12 | 2 | 9 | 2 | 66 | 5 | 44 | 4 | 33 | 4 | 27 | 3 | 22 | 3 | 19 | 3 | 12 | 2 | 9 | 2 | | 20 | 4 | 15 | 3 | 12 | 3 | 14 | 3 | 13 | 3 | 12 | 3 | 12 | 2 | 9 | 2 | |
| f4: | 111 | 173 | 55 | 5 | 37 | 4 | 28 | 3 | 22 | 3 | 19 | 3 | 16 | 3 | 10 | 2 | 8 | 2 | 55 | 5 | 37 | 4 | 28 | 3 | 21 | $\underline{4}$ | 17 | $\underline{4}$ | 16 | 3 | 10 | 2 | 8 | 2 | | 22 | 4 | 23 | 4 | 17 | 3 | 11 | 3 | 11 | 3 | 8 | 3 | 10 | 2 | 8 | 2 | |
| f5: | 78 | 102 | 39 | 4 | 26 | 4 | 20 | 3 | 16 | 3 | 13 | 3 | 11 | 3 | 7 | 2 | 6 | 2 | 39 | 4 | 26 | 4 | 18 | 3 | 12 | 3 | 10 | 3 | 11 | 3 | 7 | 2 | 6 | 2 | | 20 | 4 | 14 | 3 | 13 | 3 | 10 | 3 | 9 | 3 | 8 | 3 | 7 | 2 | 6 | 2 | |
| f6: | 111 | 173 | 55 | 5 | 37 | 4 | 28 | 3 | 22 | 3 | 19 | 3 | 16 | 3 | 10 | 2 | 8 | 2 | 55 | 5 | 37 | 4 | 28 | 3 | 21 | $\underline{4}$ | 19 | 3 | 16 | 3 | 10 | 2 | 5 | 2 | | 24 | 4 | 26 | 4 | 20 | 3 | 12 | 3 | 11 | 3 | 8 | 3 | 10 | 2 | 5 | 2 | |
| f7: | 141 | 198 | 70 | 5 | 47 | 4 | 35 | 4 | 28 | 3 | 24 | 3 | 20 | 3 | 13 | 2 | 10 | 2 | 70 | 5 | 47 | 4 | 35 | 4 | 28 | 3 | 24 | 3 | 20 | 3 | 13 | 2 | 10 | 2 | | 17 | 4 | 16 | 3 | 14 | 3 | 13 | 3 | 11 | 3 | 10 | 3 | 11 | 2 | 10 | 2 | |
| f8: | 70 | 99 | 35 | 4 | 23 | 4 | 18 | 3 | 14 | 3 | 12 | 3 | 10 | 3 | 7 | 2 | 5 | 2 | 35 | 4 | 23 | 4 | 14 | $\underline{4}$ | 12 | $\underline{4}$ | 10 | 3 | 10 | 3 | 4 | 2 | 4 | 2 | | 14 | 4 | 10 | 3 | 11 | 3 | 8 | 3 | 8 | 3 | 8 | 3 | 4 | 2 | 4 | 2 | |
| f9: | 113 | 159 | 56 | 5 | 38 | 4 | 28 | 3 | 23 | 3 | 19 | 3 | 16 | 3 | 11 | 2 | 8 | 2 | 56 | 5 | 38 | 4 | 28 | 3 | 23 | 3 | 19 | 3 | 16 | 3 | 11 | 2 | 8 | 2 | | 29 | 5 | 21 | 4 | 16 | $\underline{4}$ | 15 | 3 | 12 | 3 | 16 | 3 | 10 | $\underline{3}$ | 8 | 2 | |
| misex3.pla | | | 612 | 5 | 410 | 4 | 309 | 4 | 249 | 3 | 208 | 3 | 178 | 3 | 117 | 2 | 87 | 2 | 612 | 5 | 401 | $\underline{5}$ | 303 | 4 | 237 | $\underline{4}$ | 201 | $\underline{4}$ | 178 | 3 | 114 | 2 | 81 | 2 | 43 | 286 | 5 | 227 | 4 | 188 | 4 | 157 | 3 | 139 | 3 | 126 | 3 | 109 | $\underline{3}$ | 81 | 2 | 857 |
| | | | 3590 | | 2426 | | 1858 | | 1506 | | 1286 | | 1134 | | 751 | | 587 | | 3246 | | 2088 | | 1591 | | 1255 | | 1097 | | 997 | | 661 | | 508 | | | 2453 | | 1666 | | 1327 | | 1089 | | 941 | | 861 | | 609 | | 483 | | |
| | | | | 75 | | 75 | | 61 | | 52 | | 48 | | 47 | | 45 | | 38 | | 85 | | 85 | | 66 | | 59 | | 54 | | 53 | | 49 | | 44 | | | 83 | | 83 | | 61 | | 56 | | 53 | | 51 | | 48 | | 44 | |

representing the gain (gain $Z_{MC} = MC_{Quartus} / MC_{dekBDDE}$). This number shows how many times the number of blocks was reduced due to the use of the dekBDD + E algorithm. The last column presents the time of propagation (gain $Z_{tp} = tp_{Quartus}/tp_{dekBDDE}$). The last row of the table contains the totals of the values of the individual columns.

Among 23 benchmarks considered, in 15 cases there was observed a reduction in the number of macrocells. In the majority of cases, such a reduction was as high as several dozen percent. A surprisingly good result was reached in some cases. The best result was obtained for the spla circuit, where the number of macrocells dropped from 927 to 89 (above a tenfold decrease in the number of blocks). Similar results were obtained for pdc, f51m and rd84 benchmarks. After the Quartus II synthesis, it was impossible to develop the pdc and spla bench-

Table 2. Comparison between decompositions that employ the BDD (DekBBD, DekBBD+E) and two-stage decomposition (PALDec) referred to the classical method.

| | | k = 3 | | k = 4 | | k = 5 | | k = 6 | | k = 7 | | k = 8 | | k = 12 | | k = 16 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Benchmark | Method | B | L | B | L | B | L | B | L | B | L | B | L | B | L | B | L |
| 5xp1 | Classic | 35 | 3 | 26 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | DekBDD | 27 | 4 | 23 | 3 | 19 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | DekBDD+E | 26 | 3 | 21 | 2 | 19 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | PALDec | **26** | **3** | **21** | **2** | **18** | **2** | **17** | **2** | **16** | **2** | **14** | **2** | **12** | **2** | **11** | **2** |
| 9sym | Classic | 43 | 5 | 29 | 4 | 22 | 3 | 17 | 3 | 15 | 3 | 13 | 3 | 8 | 2 | 6 | 2 |
| | DekBDD | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 |
| | DekBDD+E | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 |
| | PALDec | 37 | 5 | 20 | 4 | 17 | 3 | **14** | **3** | 12 | 3 | **7** | **3** | **7** | **3** | **5** | **2** |
| bw | Classic | 52 | 2 | 40 | 2 | 33 | 2 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 |
| | DekBDD | 52 | 2 | 40 | 2 | 33 | 2 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 |
| | DekBDD+E | 52 | 2 | 40 | 2 | 33 | 2 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 |
| | PALDec | 52 | 2 | 40 | 2 | 33 | 2 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 | 28 | 1 |
| clip | Classic | 73 | 4 | 49 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 |
| | DekBDD | 68 | 4 | 44 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 |
| | DekBDD+E | 66 | 4 | 44 | 3 | 38 | 3 | 30 | 3 | 26 | 2 | 22 | 2 | 14 | 2 | 12 | 2 |
| | PALDec | **65** | **4** | **44** | **3** | **34** | **3** | **28** | **3** | **24** | **3** | **21** | **2** | **14** | **2** | **12** | **2** |
| f51m | Classic | 37 | 3 | 27 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 11 | 2 | 10 | 2 |
| | DekBDD | 37 | 3 | 19 | 3 | 17 | 3 | 15 | 2 | 15 | 2 | 15 | 2 | 11 | 2 | 10 | 2 |
| | DekBDD+E | 37 | 3 | 19 | 3 | 17 | 3 | 15 | 2 | 15 | 2 | 15 | 2 | 11 | 2 | 10 | 2 |
| | PALDec | **23** | **4** | **19** | **3** | **16** | **3** | **14** | **2** | **14** | **2** | **14** | **2** | **11** | **2** | **10** | **2** |
| rd53 | Classic | 15 | 3 | 10 | 2 | 8 | 2 | 6 | 2 | 6 | 2 | 6 | 2 | 4 | 2 | 3 | 1 |
| | DekBDD | 11 | 4 | 7 | 2 | 6 | 2 | 5 | 2 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 1 |
| | DekBDD+E | 11 | 4 | 7 | 2 | 6 | 2 | 5 | 2 | 5 | 2 | 5 | 2 | 4 | 2 | 3 | 1 |
| | PALDec | **11** | **4** | **7** | **2** | **6** | **2** | **5** | **2** | **5** | **2** | **5** | **2** | **4** | **2** | **3** | **1** |
| rd73 | Classic | 70 | 4 | 47 | 3 | 36 | 3 | 29 | 3 | 24 | 3 | 20 | 2 | 14 | 2 | 11 | 2 |
| | DekBDD | 42 | 6 | 18 | 3 | 16 | 3 | 14 | 3 | 13 | 3 | 10 | 2 | 9 | 2 | 8 | 2 |
| | DekBDD+E | 42 | 6 | 18 | 3 | 16 | 3 | 13 | 3 | 13 | 3 | 10 | 2 | 9 | 2 | 8 | 2 |
| | PALDec | **42** | **5** | **18** | **3** | **16** | **3** | **13** | **3** | **13** | **3** | **10** | **2** | **9** | **2** | **8** | **2** |
| rd84 | Classic | 142 | 5 | 95 | 4 | 72 | 4 | 58 | 3 | 49 | 3 | 42 | 3 | 28 | 2 | 21 | 2 |
| | DekBDD | 77 | 7 | 27 | 4 | 23 | 4 | 21 | 4 | 20 | 4 | 15 | 3 | 12 | 3 | 11 | 2 |
| | DekBDD+E | 75 | 7 | 27 | 4 | 23 | 4 | 21 | 4 | 20 | 4 | 15 | 3 | 12 | 3 | 11 | 2 |
| | PALDec | **75** | **7** | **27** | **4** | **23** | **3** | **23** | **3** | **22** | **3** | **15** | **3** | **12** | **3** | **10** | **2** |
| sao2 | Classic | 36 | 3 | 24 | 3 | 19 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 |
| | DekBDD | 29 | 4 | 23 | 3 | 18 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 |
| | DekBDD+E | 26 | 3 | 20 | 3 | 18 | 2 | 15 | 2 | 14 | 2 | 11 | 2 | 7 | 2 | 7 | 2 |
| | PALDec | **26** | **4** | **19** | **3** | **17** | **3** | **13** | **2** | **13** | **2** | **11** | **2** | **7** | **2** | **7** | **2** |
| xor5 | Classic | 8 | 3 | 5 | 2 | 4 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 1 | 1 |
| | DekBDD | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| | DekBDD+E | 4 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| | PALDec | **4** | **4** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **1** | **1** |
| Misex1 | Classic | 14 | 2 | 12 | 2 | 8 | 2 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 |
| | DekBDD | 14 | 2 | 12 | 2 | 8 | 2 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 |
| | DekBDD+E | 14 | 2 | 12 | 2 | 8 | 2 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 |
| | PALDec | 14 | 2 | 12 | 2 | 8 | 2 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 | 7 | 1 |
| Misex2 | Classic | 19 | 2 | 19 | 2 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 |
| | DekBDD | 19 | 2 | 19 | 2 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 |
| | DekBDD+E | 19 | 2 | 19 | 2 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 |
| | PALDec | 19 | 2 | 19 | 2 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 | 18 | 1 |
| apex4 | Classic | 494 | 4 | 333 | 4 | 249 | 3 | 204 | 3 | 170 | 3 | 148 | 3 | 97 | 2 | 73 | 2 |
| | DekBDD | 494 | 4 | 333 | 4 | 249 | 3 | 204 | 3 | 170 | 3 | 148 | 3 | 97 | 2 | 73 | 2 |
| | DekBDD+E | 493 | 5 | 333 | 4 | 249 | 3 | 204 | 3 | 170 | 3 | 148 | 3 | 97 | 2 | 73 | 2 |
| | PALDec | 494 | 4 | 333 | 4 | 249 | 3 | 203 | 3 | 170 | 3 | 148 | 3 | 97 | 2 | 73 | 2 |
| squar 5 | Classic | 14 | 2 | 11 | 2 | 9 | 2 | 9 | 2 | 9 | 2 | 8 | 1 | 8 | 1 | 8 | 1 |
| | DekBDD | 14 | 2 | 11 | 2 | 9 | 2 | 9 | 2 | 9 | 2 | 8 | 1 | 8 | 1 | 8 | 1 |
| | DekBDD+E | 13 | 2 | 11 | 2 | 9 | 2 | 9 | 2 | 9 | 2 | 8 | 1 | 8 | 1 | 8 | 1 |
| | PALDec | 13 | 2 | 11 | 2 | 9 | 2 | 9 | 2 | 9 | 2 | 8 | 1 | 8 | 1 | 8 | 1 |
| z5xp1 | Classic | 35 | 3 | 26 | 3 | 22 | 2 | 18 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | DekBDD | 35 | 3 | 21 | 2 | 19 | 2 | 17 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | DekBDD+E | 35 | 3 | 21 | 2 | 19 | 2 | 17 | 2 | 16 | 2 | 15 | 2 | 12 | 2 | 11 | 2 |
| | PALDec | **26** | **3** | **21** | **2** | **18** | **2** | **17** | **2** | **16** | **2** | **14** | **2** | **12** | **2** | **11** | **2** |
| z9sym | Classic | 43 | 5 | 29 | 4 | 22 | 3 | 17 | 3 | 15 | 3 | 13 | 3 | 8 | 2 | 6 | 2 |
| | DekBDD | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 |
| | DekBDD+E | 37 | 5 | 20 | 4 | 17 | 3 | 14 | 3 | 12 | 3 | 7 | 3 | 7 | 3 | 5 | 2 |
| | PALDec | 37 | 5 | 20 | 4 | 17 | 3 | **14** | **3** | 12 | 3 | **7** | **3** | **7** | **3** | **5** | **2** |
| | Classic | 1130 | 53 | 782 | 46 | 604 | 38 | 495 | 35 | 432 | 34 | 384 | 32 | 278 | 28 | 233 | 26 |
| | DekBDD | 997 | 61 | 639 | 45 | 509 | 39 | 431 | 36 | 383 | 35 | 333 | 32 | 255 | 31 | 218 | 26 |
| | DekBDD+E | 987 | 60 | 634 | 44 | 509 | 39 | 430 | 36 | 383 | 35 | 333 | 32 | 255 | 31 | 218 | 26 |
| | PALDec | 964 | 60 | 633 | 44 | 501 | 39 | 425 | 35 | 381 | 35 | 329 | 32 | 255 | 31 | 217 | 26 |

Table 3. Comparison between the number of macrocells (MAX 7000B) after synthesis using Quartus II and dekBDD+E.

| | PAL blocks k=5 | | MAX 7000B | | | MAX 7000B | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Classic | BDD+E | Quartus II: area opt. | | | dekBDD + E+Quartus II | | | Gain | Gain |
| Benchmark | B | B | MC | Exp | tp [ns] | MC | Exp | tp [ns] | $Z_{MC}$ | $Z_{tp}$ |
| 5xp1.pla | 22 | 19 | 17 | 8 | 11,5 | **16** | 6 | **11,5** | 1,06 | 1,00 |
| 9sym.pla | 18 | 17 | 19 | 3 | 15,8 | **15** | 13 | 18,4 | 1,27 | 0,86 |
| clip.pla | 38 | 38 | 24 | 11 | 16 | 27 | 10 | **14,5** | 0,89 | 1,10 |
| f51m.pla | 22 | 17 | 96 | 56 | 15,6 | **15** | 3 | **11,3** | 6,40 | 1,38 |
| rd53.pla | 8 | 6 | 7 | 2 | 11,3 | **4** | 10 | **9,8** | 1,75 | 1,15 |
| rd73.pla | 36 | 16 | 23 | 12 | 20,1 | **12** | 11 | **15,6** | 1,92 | 1,29 |
| rd84.pla | 67 | 21 | 53 | 18 | 16 | **15** | 12 | 18,4 | 3,53 | 0,87 |
| sao2.pla | 14 | 14 | 15 | 3 | 14,1 | **12** | 4 | **14,1** | 1,25 | 1,00 |
| xor5.pla | 4 | 2 | 4 | 0 | 9,8 | **2** | 0 | **9,8** | 2,00 | 1,00 |
| z5xp1.pla | 22 | 19 | 18 | 10 | 14,3 | **16** | 6 | **11,5** | 1,13 | 1,24 |
| z9sym.pla | 18 | 17 | 19 | 13 | 18,6 | **15** | 13 | **18,4** | 1,27 | 1,01 |
| alu4.pla | 126 | 125 | 159 | 48 | 27,3 | **102** | 31 | **22,6** | 1,56 | 1,21 |
| cordic.pla | 73 | 42 | 13 | 6 | 18,6 | 28 | 10 | 31,6 | 0,46 | 0,59 |
| Apex5.pla | 299 | 289 | 132 | 107 | 18,2 | 141 | 124 | 18,9 | 0,94 | 0,96 |
| t481.pla | 90 | 32 | 4 | 13 | 11,3 | 23 | 4 | 28,5 | 0,17 | 0,40 |
| misex3.pla | 122 | 122 | 169 | 75 | 26,1 | **101** | 35 | **20,7** | 1,67 | 1,26 |
| bw.pla | 33 | 33 | 28 | 5 | 7,5 | **28** | 6 | **7,5** | 1,00 | 1,00 |
| misex1.pla | 8 | 8 | 8 | 0 | 10 | **7** | 1 | **7,2** | 1,14 | 1,39 |
| misex2.pla | 18 | 18 | 18 | 0 | 5,8 | **18** | 0 | **5,8** | 1,00 | 1,00 |
| squar5.pla | 9 | 9 | 8 | 4 | 7,2 | 9 | 0 | 10 | 0,89 | 0,72 |
| Apex4.pla | 246 | 246 | 244 | 124 | 20,6 | 246 | 63 | 22,1 | 0,99 | 0,93 |
| pdc.pla | 124 | 121 | 871 | 802 | - | **92** | 51 | **16,6** | 9,47 | |
| spla.pla | 125 | 122 | 927 | 1056 | - | **89** | 54 | **16,7** | 10,42 | |
| Sum | 1542 | 1353 | 2876 | 2376 | 315,7 | **1033** | **467** | 328,2 | **2,8** | 0,96 |

marks using even MAX7000B with the highest number of macrocells.

The benchmark t481, for which the considerable growth of the number of macrocells was observed, is a special case (from four to 23). This benchmark requires a different strategy of decomposition. Quartus II allows the description of a circuit to be created, where several outputs of macrocells can be connected to one product-term. The reason for a large difference in the number of blocks for t481 is because the dekBDD + E algorithm allows, at most, to connect one macrocell output to one product. A suitable modification of the dekBDD + E algorithm, which also takes into account such cases, will be the subject of further studies.

Despite the unfavourable result for one benchmark, the total number of macrocells for all benchmarks was reduced almost three times (2.8 times). Also, in the majority of cases, the number of shared expanders was reduced proportionally to the reduction in the number of macrocells. It is worth noticing that the proposed description in the Verilog language does not exclude the use of expanders in the firmware synthesis tool. For example, in the case of the 5xp1 benchmark, the dekBDD + E method enables one to create the description using 19 PAL-based blocks ($k = 5$). Once the final stage of the synthesis using the firmware tool is made, 16 MAX 7000B macrocells are created based on the description of 19 blocks (developed around the PAL-type core) and six expanders. Thus, the use of the firmware tool enabled us to take advantage of the specific features of the architecture of a programmable structure and to further improve the decomposition results as well.

In the majority of cases, no increase in the propagation time was observed while reducing the number of blocks (16 of 23 benchmarks). The average gain of propagation time remained almost unchanged (difference of 4%). However, this average value does not take into account the fact that the number of macrocells used in the firmware tool was higher than the maximum available number of macrocells in the circuits of the MAX7000B for two benchmarks. On the other hand, the use of the dekBDD + E algorithm enabled us to build the same circuits and to obtain propagation times as low as 17 ns. This confirms the practical effectiveness of the proposed methods and of their description in HDLs.

Focus should also be put on the working speed of the proposed algorithms and the influence that processing of the circuit description files has on the duration of the synthesis with the use of firmware tools. The average synthesis duration was 10 and 12 minutes, respectively, for pdc and spla using Quartus II. In the case of the proposed dekBDD+E module, the synthesis of the same circuits using Quartus II took 30 seconds. Thus, as high as a

20-fold speed-up of the whole synthesis process was obtained.

## 5. Conclusions

The paper presents a logic synthesis method dedicated to PAL-based CPLDs. The aim of that method was to utilize non-standard decomposition in order to minimize the area of the implemented circuit and the reduction of necessary logic blocks in the programmable structure. These methods provide an alternative to the classical approach based on two-level minimization of individual single-output functions.

The paper presents three variants of PAL-oriented decomposition dedicated to PAL-based CPLDs. First, two-stage PAL-oriented decomposition is presented. This method is an extension of the classical Ashenhurst-Curtis decomposition. Decomposition based on a two-stage model is very effective. Unfortunately, the algorithms contain very demanding procedures. The computation complexity precludes the synthesis of large designs. Other PAL-oriented decomposition models use reduced ordered binary decision diagrams. The binary decision diagram was taken into consideration in order to increase computation performance/efficiency. The experience gained in the implementation of two-stage decomposition allows us to implement efficient partitioning procedures for the BDD. Decomposition results for BDD methods are slightly worse as referenced to previous approaches. The synthesis process is computation efficient and allows us to decompose complex logic circuits in a reasonable amount of time. The exploration of BDD decomposition methods shows their undiscovered potential; a potential which can still be developed, especially for the decomposition of a function consisting of a few hundred input and output variables.

The essence of all the methods is to incorporate decomposition into the synthesis process dedicated to CPLD structures. The algorithm consists in a sequential search for decomposition which provides the feasibility of implementation of a free block in one PAL-based logic block containing a predefined number of product terms.

The proposed methods were practically proved. For all synthesis methods, the results of the experiments presented in the paper become close to one another with growing $k$. The conclusion is that for large $k$ it is better to use the dekBDD+E approach, which works fast and gives comparable results.

Through the adjustment of the decomposition elements to the logical resources characteristic for a PAL-based logic block, a significant improvement of the synthesis effectiveness in relation to the classical approach could be obtained. Unfortunately, a reduction in the area is not always associated with a reduction in logic levels.

Although satisfactory results were achieved, the presented methods will still be improved. In our opinion, the quality of results could be enhanced, e.g., by extending the decomposition model to allow creating the description of a circuit, where several outputs of macrocells can be connected to one product-term. Considering perspectives for further research, comparisons with other tools and integration with commercial tools will be taken into consideration.

## References

Akers, S.B. (1978). Functional testing with binary decision diagrams, *Proceedings of the 8-th Annual Conference on Fault-Tolerant Computing*, pp. 75–82.

Anderson, J.H. and Brown, S.D. (1998). Technology mapping for large complex PLDs, *DAC '98: Proceedings of the 35th Annual Design Automation Conference, New York, NY, USA*, pp. 698–703.

Ashar, P., Devadas, S. and Newton, A.R. (1992). *Sequential Logic Synthesis*, Kluwer Academic Publishers, Norwell, MA.

Ashenhurst, R. (1957). The decomposition of switching functions, *Proceedings of an International Symposium on the Theory of Switching*, pp. 74–116.

Bolton, M. (1990). *Digital Systems Design with Programmable Logic*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Brace, K.S., Rudell, R.L. and Bryant, R.E. (1990). Efficient implementation of a bdd package, *DAC '90: Proceedings of the 27th ACM/IEEE Design Automation Conference, New York, NY, USA*, pp. 40–45.

Brayton, R.K., Sangiovanni-Vincentelli, A.L., McMullen, C.T. and Hachtel, G. D. (1984). *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Norwell, MA.

Bryant, R.E. (1986). Graph-based algorithms for Boolean function manipulation, *IEEE Transactions on Computers* **35**(8): 677–691.

Burns, M., Perkowski, M., Jóźwiak, L. and Grygiel, S. (1998). An efficient and effective approach to column-based input/output encoding in functional decomposition, *Proceedings of the 3rd International Workshop on Boolean Problems, Freiberg, Germany*, pp. 19–29.

Chartrand, G. and Zhang, P. (2008). *Chromatic Graph Theory*, Chapman & Hall/CRC, Boca Raton, FL.

Chen, K. and Muroga, S. (1988). Input assignment algorithm for decoded-PLAs with multi-input decoders, *IEEE International Conference on Computer-Aided Design. ICCAD-88. Digest of Technical Papers, Santa Clara, CA, USA*, pp. 474–477.

Chen, S., Hwang, T. and Liu, C. (2002). A technology mapping algorithm for CPLD architectures, *2002 IEEE International Conference on Field-Programmable Technology*, pp. 204–210.

Ciesielski, M. and Yang, S. (1992). PLADE: A two-stage PLA decomposition, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **11**(8): 943–954.

Curtis, H. (1962). *A New Approach to the Design of Switching Circuits*, Van Nostrand, Princeton, NJ.

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

Devadas, S., Wang, A., Newton, A. and Sangiovanni-Vincentelli, A. (1988). Boolean decomposition of programmable logic arrays, *Proceedings of the IEEE Custom Integrated Circuit Conference, Rochester, NY, USA*, Vol. 2, pp. 2.5.1–2.5.5.

Ebendt, R., Fey, G. and Drechsler, R. (2005). *Advanced BDD Optimization*, Springer-Verlag, Berlin/Heidelberg.

Kania, D. (2004). *Logic Synthesis for PAL-based Complex Programmable Logic Devices*, Zeszyty Naukowe: Elektronika, Vol. 14, pp. 5–212, (in Polish).

Kania, D., Kulisz, J. and Milik, A. (2005). A novel method of two-stage decomposition dedicated for PAL-based CPLDs, *Proceedings of the 8th Euromicro Conference on Digital System Design, Porto, Portugal*, pp. 114–121.

Kim, J., Kim, H. and Lin, C. (2001). A new technology mapping for CPLD under the time constraint, *Proceedings of the Asia and South Pacific Design Automation Conference, Yokohama, Japan*, pp. 235–238.

Kouloheris, J. and Gamal, A. (1992). PLA-based FPGA area versus cell C+ granularity, *Proceedings of the Custom Integrated Circuits Conference, Boston, MA, USA*, Vol. 4, pp. 4.3.1–4.3.4.

Lai, Y., Pan, K. and Pedram, M. (1994). FPGA synthesis using function decomposition, *ICCS '94: Proceedings of the 1994 IEEE International Conference on Computer Design: VLSI in Computer & Processors, Washington, DC, USA*, pp. 30–35.

Lai, Y., Pan, K. and Pedram, M. (1996). OBDD-based function decomposition: Algorithms and implementation, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **15**(8): 977–990.

Minato, S. (1996). *Binary Decision Diagrams and Applications for VLSI CAD*, Kluwer Academic Publishers, Norwell, MA.

Murgai, R., Brayton, R. and Sangiovanni-Vincentelli, A. (1994). Optimum functional decomposition using encoding, *Proceedings of the 31st Annual Conference on Design Automation, New York, NY, USA*, pp. 408–414.

Muthukumar, V. (2001). An improved input-output encoding approach for functional decomposition, *DSD '01: Proceedings of the Euromicro Symposium on Digital Systems Design, Washington, DC, USA*, pp. 144–147.

Muthukumar, V., Bignall, R. and Selvaraj, H. (2000). An input-output encoding approach for serial decomposition, *SBCCI'00: Proceedings of the 13th Symposium on Integrated Circuits and Systems Design, Washington, DC, USA*, pp. 61–68.

Nowicka, M., Łuba, T. and Selvaraj, H. (1997). Multilevel decomposition strategies in decomposition-based algorithms and tools, *International Workshop on Logic and Architecture Synthesis, Grenoble, France*, pp. 129–136.

Opara, A. (2009). *Decompositional Synthesis Methods of Combinatorial Circuits with Binary Decision Diagrams Application*, Ph.D. thesis, Silesian University of Technology, Gliwice, (in Polish).

Opara, A. and Kania, D. (2009). A novel non-disjunctive method for decomposition of CPLDs, *Electronics and Telecommunications Quarterly* **55**(1): 95–111.

Rawski, M., Łuba, T. and Falkowski, B. (2008). Logic synthesis method for FPGAs with embedded memory blocks, *IEEE International Symposium on Circuits and Systems, Seattle, WA, USA*, pp. 2014–2017.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams, *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, Los Alamitos, CA, USA,* pp. 42–47.

Saldanha, A., Villa, T., Brayton, R. and Sangiovanni-Vincentelli, A. (1994). Satisfaction of input and output encoding constraints, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **13**(5): 589–602.

Scholl, C. (2001). *Functional Decomposition with Application to FPGA Synthesis*, Kluwer Academic Publishers, Norwell, MA.

Yan, K. (2001). Practical logic synthesis for CPLDs and FPGAs with PLA-style logic blocks, *Proceedings of the 2001 Conference on Asia South Pacific Design Automation, ACM New York, NY, USA*, pp. 231–234.

Yang, C. and Ciesielski, M. (2002). BDS: A BDD-based logic optimization system, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **21**(7): 866–876.

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

**Adam Opara** received the M.Sc. degree in computer science in 2002 from the Silesian University of Technology, Gliwice, Poland, and the Ph.D. degree in 2009. Since 2009 he has been an assistant professor at the Institute of Computer Science of the Silesian University of Technology. His research interest include hardware description languages, logic synthesis, programmable digital circuits and systems.

**Dariusz Kania** received the 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, Gliwice. His main interests and research areas include programmable devices and systems, logic synthesis and optimization dedicated to a wide range of programmable logic devices (CPLD, FPGA) and implementation of digital circuits.

Received: 13 July 2009
Revised: 16 January 2010