FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 31, Nº 2, June 2018, pp. 223 - 240
https://doi.org/10.2298/FUEE1802223S

COMPACT XOR-BI-DECOMPOSITION FOR LATTICES OF BOOLEAN FUNCTIONS*

Bernd Steinbach[1], Christian Posthoff[2]

[1]Freiberg University of Mining and Technology, Institute of Computer Science, Freiberg, Germany
[2]The University of the West Indies, Department of Computing and Information Technology, Saint Augustine, Trinidad and Tobago

Abstract. *Bi-Decomposition is a powerful approach for the synthesis of multi-level combinational circuits because it utilizes the properties of the given functions to find small circuits, with low power consumption and low delay. Compact bi-decompositions restrict the variables in the support of the decomposition functions as much as possible. Methods to find compact AND-, OR-, or XOR-bi-decompositions for a given completely specified function are well known.*
A lattice of Boolean functions represents all possible functions which are defined by an incompletely specified function. Lattices of Boolean Functions significantly increase the possibilities to synthesize a minimal circuit. However, so far only methods to find compact AND- or OR-bi-decompositions for lattices of Boolean functions are known. This gap, i.e., a method to find a compact XOR-bi-decomposition for a lattice of Boolean functions, has been closed by the approach suggested in this paper.

Key words: *synthesis, combinational circuit, lattice of Boolean functions, XOR-bidecomposition, Boolean Differential Calculus, derivative operations.*

1 INTRODUCTION

The aim of all decomposition methods in circuit design is to find decomposition functions that are simpler than the given function. The bi-decomposition is an approach that decomposes a given Boolean function into two simpler decomposition functions which are combined by an AND-, an OR-, or an XOR-gate. For the aim of simplification, the bi-decomposition utilizes the properties of the given Boolean function to design circuit structures of a small area, low power consumption, and low delay [1].

Received October 21, 2017; received in revised form January 24, 2018
Corresponding author: Bernd Steinbach
Institute of Computer Science, Freiberg University of Mining and Technology, Bernhard-von-Cotta-Str. 2, D-09596 Freiberg, Germany
(E-mail: steinb@informatik.tu-freiberg.de)
*An earlier version of this paper was presented as an invited address at the Reed-Muller 2017 Workshop, Novi Sad, Serbia, May 24-25, 2017

There are several types of bi-decompositions. Both decomposition functions of each strong bi-decomposition are simpler than the given function because they depend on fewer variables. Unfortunately, there are functions for which no strong bi-decomposition exists. Le [2] bridged this gap by means of the weak bi-decomposition. He found that each function, for which neither a weak OR bi-decomposition nor a weak AND bi-decomposition exists, can be simplified by a strong XOR bi-decomposition. A simplified proof of the completeness of the strong and weak bi-decomposition is given in [3, 4]. An implementation that reuses already decomposed blocks outperforms other synthesis approaches [5]. A drawback of the weak bi-decomposition is that the synthesized circuits can have a large difference between the shortest and the longest path (unbalanced circuits).

Recently, vectorial bi-decompositions were suggested as supplement to strong and weak bi-decompositions [6, 7]. The decomposition functions of these bi-decompositions are simpler than the given function because they are independent of the simultaneous change of several variables. Vectorial bi-decompositions can exist for functions without any strong bi-decomposition. Benefits of the vectorial bi-decomposition are their contribution to balanced circuits and the increased number of decomposition possibilities in comparison to the strong bi-decomposition. All bi-decomposition approaches mentioned above utilize the Boolean Differential Calculus (BDC) [3, 4, 8–12] to find optimal bi-decompositions.

There are several other approaches of bi-decompositions which demonstrate the interest on this useful synthesis method; however, these other approaches are not directly helpful to solve the problem explored in this paper. In [13] a method for disjoint bi-decompositions with an extension to non-disjoint bi-decompositions for a single common variable are suggested. A graph-based approach for bi-decompositions was suggested in [14]. Unfortunately, the used benchmarks in [5] and [14] overlap only partially. One common benchmark is t481 where our approach from [5] outperforms the graph-based approach from [14] in the number of gates (17/25). Recently, semi-tensor products of matrices were suggested for bi-decompositions of Boolean and multi-valued functions [15]. However, this paper does not contain experimental results of benchmark circuits.

It is a property of the given function whether it can be decomposed into two simpler decomposition functions using a certain type of bi-decomposition. The possibility to find a bi-decomposition increases when the function to decompose can be chosen from a lattice of Boolean functions. Incompletely specified functions were traditionally used as a source of a lattice of Boolean functions. The ON-set-function $f_q(\mathbf{x})$ and the OFF-set-function $f_r(\mathbf{x})$ are the preferred mark functions of these lattices. The introduction of derivative operations for lattices of Boolean functions [4, 16, 17] facilitates the application of lattices in circuit design.

A strong bi-decomposition divides the variables of the function to decompose into three disjoint subsets. The variables $\mathbf{x}_a$ control only the decomposition function g, the variables $\mathbf{x}_b$ control only the decomposition function h, and the variables $\mathbf{x}_c$ are commonly used for both the decomposition functions g and h. The more variables in the dedicated sets of variables $\mathbf{x}_a$ and $\mathbf{x}_b$ the simpler are the decomposition functions g and h. A *compact* strong bi-decomposition uses the largest possible sets of $\mathbf{x}_a$ and $\mathbf{x}_b$.

There are formulas [3, 4, 10–12] containing operations of the Boolean Differential Calculus that can be used to decide whether a lattice of Boolean functions contains a compact strong AND- or a compact strong OR-bi-decomposition. Unfortunately, so far it is only possible to find a compact strong XOR-bi-decomposition for a single decomposition function or to assign only a single variable to the set $\mathbf{x}_a$ for the check whether a lattice of Boolean functions contains a strong XOR-bi-decomposition [18]. We suggest in this paper an approach to find also compact strong XOR-bi-decompositions for a lattice of Boolean functions. This new method combines the ideas of simplifications used in both the strong and the vectorial bi-decomposition. Hence, we are going to solve a problem that is more than 20 years known as unsolved.

The rest of this paper is organized as follows. Section 2 briefly describes lattices of Boolean functions and single derivatives of functions belonging to such a lattice. Section 3 summarizes the known approach to find and determine non-compact XOR-bi-decompositions. Section 4 introduces into the theory of compact XOR-bi-bi-decompositions, concludes the main theorem and the consequence for compact XOR-bi-bi-decompositions, and provides two consecutive algorithms that solve this task using XBOOLE [19, 20]. Section 5 demonstrates the benefits of the suggested new decomposition method by means of a simple example. Section 6 concludes the paper.

2 Lattices of Boolean Functions

Lattices of Boolean functions occur, e.g., in circuit design where each function of the lattice can be chosen as a function to realize the circuit structure. Hence, lattices of Boolean functions provide a possibility for optimization in circuit design.

Widely used are lattices which can be modeled as incompletely specified function (ISF). Such an incompletely specified Boolean function divides the 2^n patterns

$\mathbf{x}$ of the Boolean space $\mathbb{B}^n$ into three disjoint sets:

$$\begin{aligned}
\mathbf{x} \in \text{ don't-care-set} &\Leftrightarrow f_\varphi(x_1,\ldots,x_n) = 1 \\
&\Leftrightarrow \text{it is allowed to choose the function value of} \\
&\quad f(\mathbf{x}) \text{ without any restrictions;} \\
\mathbf{x} \in \text{ ON-set} &\Leftrightarrow f_q(x_1,\ldots,x_n) = 1 \\
&\Leftrightarrow (f_\varphi(x_1,\ldots,x_n) = 0) \wedge (f(x_1,\ldots,x_n) = 1)\ ; \\
\mathbf{x} \in \text{ OFF-set} &\Leftrightarrow f_r(x_1,\ldots,x_n) = 1 \\
&\Leftrightarrow (f_\varphi(x_1,\ldots,x_n) = 0) \wedge (f(x_1,\ldots,x_n) = 0)\ .
\end{aligned}$$

Each pair of these mark functions can be used to specify all functions of the lattice. A function $f(\mathbf{x})$ belongs to the lattice $L\langle f_q(\mathbf{x}), f_r(\mathbf{x})\rangle$ if

$$f_q(\mathbf{x}) \le f(\mathbf{x}) \le \overline{f_r(\mathbf{x})}\ .$$

The single derivatives with regard to x_i of all functions of a lattice $L\langle f_q(\mathbf{x}), f_r(\mathbf{x})\rangle$ results again in a lattice of Boolean function that is specified by the mark functions:

$$f_q^{\partial x_i}(\mathbf{x}_1) = \max_{x_i} f_q(x_i,\mathbf{x}_1) \wedge \max_{x_i} f_r(x_i,\mathbf{x}_1)\ , \tag{1}$$

$$f_r^{\partial x_i}(\mathbf{x}_1) = \min_{x_i} f_q(x_i,\mathbf{x}_1) \vee \min_{x_i} f_r(x_i,\mathbf{x}_1)\ . \tag{2}$$

3 Known Non-Compact XOR-Bi-Decompositions for Lattices of Boolean Functions

A lattice of Boolean functions $L\langle f_q(x_a,\mathbf{x}_b,\mathbf{x}_c), f_r(x_a,\mathbf{x}_b,\mathbf{x}_c)\rangle$ contains at least one function $f(x_a,\mathbf{x}_b,\mathbf{x}_c)$ that is strongly XOR-bi-decomposable with regard to the single variable x_a and the set of variables $\mathbf{x}_b$ if and only if

$$\max_{\mathbf{x}_b}{}^m f_q^{\partial x_a}(\mathbf{x}_b,\mathbf{x}_c) \wedge f_r^{\partial x_a}(\mathbf{x}_b,\mathbf{x}_c) = 0\ . \tag{3}$$

The decomposition function $g(x_a,\mathbf{x}_c)$ of this XOR-bi-decomposition is uniquely specified by

$$g(x_a,\mathbf{x}_c) = x_a \wedge \max_{\mathbf{x}_b}{}^m f_q^{\partial x_a}(\mathbf{x}_b,\mathbf{x}_c)\ , \tag{4}$$

and the associated decomposition function $h(\mathbf{x}_b,\mathbf{x}_c)$ can be chosen from the lattice with the mark functions

$$h_q(\mathbf{x}_b,\mathbf{x}_c) = \max_{x_a}(\,(\overline{g(x_a,\mathbf{x}_c)} \wedge f_q(x_a,\mathbf{x}_b,\mathbf{x}_c)) \vee (g(x_a,\mathbf{x}_c) \wedge f_r(x_a,\mathbf{x}_b,\mathbf{x}_c))\,)\ , \tag{5}$$

$$h_r(\mathbf{x}_b,\mathbf{x}_c) = \max_{x_a}(\,(\overline{g(x_a,\mathbf{x}_c)} \wedge f_r(x_a,\mathbf{x}_b,\mathbf{x}_c)) \vee (g(x_a,\mathbf{x}_c) \wedge f_q(x_a,\mathbf{x}_b,\mathbf{x}_c))\,)\ . \tag{6}$$

More details about strong and weak bi-decompositions are given in [3, 4, 10, 11].

4 Compact XOR-Bi-Decomposition for Lattices of Boolean Functions

A compact bi-decomposition is determined by maximal numbers of variables in the dedicated sets $\mathbf{x}_a$ and $\mathbf{x}_b$. The number of commonly used variables $\mathbf{x}_c$ is as small as possible, and consequently the decomposition functions $g(\mathbf{x}_a, \mathbf{x}_c)$ and $h(\mathbf{x}_b, \mathbf{x}_c)$ will be the simplest in case of a desirable compact bi-decomposition.

Condition (3) allows us to find a maximal number of possible variables for the dedicated set $\mathbf{x}_b$ of an XOR-bi-decomposition, but it unfortunately restricts to a single variable x_a of the dedicated set $\mathbf{x}_a$. As initial solution we can calculate the decomposition function $g(x_a, \mathbf{x}_c)$ using (4) and the lattice $L\left\langle h_q(\mathbf{x}_b, \mathbf{x}_c), h_r(\mathbf{x}_b, \mathbf{x}_c)\right\rangle$, using (5) and (6), from which the decomposition function $h(\mathbf{x}_b, \mathbf{x}_c)$ can be chosen. The set of all variables $\mathbf{x}$ is distributed to the three disjoint sets $\mathbf{x}_a = x_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$.

We assume that $\mathbf{x}_b$ contains as much as possible variables, because it can be verified by (3) that no other variable can be added to the dedicated set $\mathbf{x}_b$ without loosing the property of an XOR-bi-decomposition of the given lattice. A given XOR-bi-decomposition is not compact if at least one variable can be moved from $\mathbf{x}_c$ to $\mathbf{x}_a$. Moving a variable x_i from $\mathbf{x}_c$ to $\mathbf{x}_a$ does not change the set of variables the function $g(\mathbf{x}_a, \mathbf{x}_c)$ is depending on; however, it reduces the support of the function $h(\mathbf{x}_b, \mathbf{x}_c)$. The set of variables $\mathbf{x}_c$ is split into x_i and $\mathbf{x}_{c0}$.

Due to the evaluation of Condition (3) for all variables, we know that the function $h(\mathbf{x}_b, \mathbf{x}_c)$ depends on all variables $(\mathbf{x}_b, \mathbf{x}_c)$. Hence, only another function $h'(\mathbf{x}_b, \mathbf{x}_{c0})$ is able to solve the problem. In the context of the XOR-bi-decomposition, the following transformation steps show the key idea to solve the problem:

$$
\begin{aligned}
& g(\mathbf{x}_{a0}, \mathbf{x}_c) \oplus h(\mathbf{x}_b, \mathbf{x}_c) && (7)\\
&= g(\mathbf{x}_{a0}, \mathbf{x}_{c0}, x_i) \oplus h(\mathbf{x}_b, \mathbf{x}_{c0}, x_i) && (8)\\
&= g(\mathbf{x}_{a0}, \mathbf{x}_{c0}, x_i) \oplus (x_i \oplus h'(\mathbf{x}_b, \mathbf{x}_{c0})) && (9)\\
&= (g(\mathbf{x}_{a0}, \mathbf{x}_{c0}, x_i) \oplus x_i) \oplus h'(\mathbf{x}_b, \mathbf{x}_{c0}) && (10)\\
&= g'(\mathbf{x}_{a0}, x_i, \mathbf{x}_{c0}) \oplus h'(\mathbf{x}_b, \mathbf{x}_{c0}) && (11)\\
&= g'(\mathbf{x}_a, \mathbf{x}_{c0}) \oplus h'(\mathbf{x}_b, \mathbf{x}_{c0})\ ; && (12)
\end{aligned}
$$

- the step from (7) to (8) emphasizes the variable x_i as element of the given set of variables $\mathbf{x}_c$;
- the step from (8) to (9) requires that the function $h(\mathbf{x}_b, \mathbf{x}_{c0}, x_i)$ is linear in x_i. This property enables or prohibits the whole transformation;
- the step from (9) to (10) moves the variable x_i to the other decomposition function of the XOR-bi-decomposition;

- the step from (10) to (11) includes the variable x_i into the new decomposition function $g'(\mathbf{x}_{a0}, x_i, \mathbf{x}_{c0})$. This transformation is possible without any restriction;
- the step from (11) to (12) emphasizes that x_i does not belong to the commonly used variables $\mathbf{x}_{c0}$, because $h'(\mathbf{x}_b, \mathbf{x}_{c0})$ does not depend on x_i; hence, x_i extends the dedicated set of variables $\mathbf{x}_{a0}$ to $\mathbf{x}_a = (\mathbf{x}_a, x_i)$.

The only condition for the transformation from (7) to (12) is that the lattice $L\langle h_q(\mathbf{x}_b, \mathbf{x}_c), h_r(\mathbf{x}_b, \mathbf{x}_c)\rangle$ contains at least one function that satisfies:

$$\frac{\partial h(\mathbf{x}_1, x_i)}{\partial x_i} = 1 \ .$$

Theorem 1 (Linear Separation of a Variable for a Function of a Lattice)
A lattice $L\langle h_q(\mathbf{x}_b, \mathbf{x}_c), h_r(\mathbf{x}_b, \mathbf{x}_c)\rangle$ contains at least one function $h(\mathbf{x}_1, x_i)$ that can be represented by

$$h(\mathbf{x}_1, x_i) = x_i \oplus h'(\mathbf{x}_1) \tag{13}$$

if and only if the condition

$$h_r^{\partial x_i}(\mathbf{x}_1) = 0 \tag{14}$$

is satisfied.

Proof 1

necessary: *Due to Condition* (14) *the lattice $L\langle h_q(\mathbf{x}_b, \mathbf{x}_c), h_r(\mathbf{x}_b, \mathbf{x}_c)\rangle$ contains at least one function that satisfies*

$$\frac{\partial h(\mathbf{x}_1, x_i)}{\partial x_i} = 1 \ . \tag{15}$$

It is well-known that (15) *is satisfied if the function $h(\mathbf{x}_1, x_i)$ is linear with regard to the variable x_i as shown in* (13)*. Hence, we have Theorem 1 in the direction* (14) $\Rightarrow$ (13)*.*

sufficient: *Function $h(\mathbf{x}_1, x_i)$* (13) *belongs to the lattice $L\langle h_q(\mathbf{x}_b, \mathbf{x}_c), h_r(\mathbf{x}_b, \mathbf{x}_c)\rangle$ so that it holds:*

$$h_q(\mathbf{x}_b, \mathbf{x}_c) \leq h(\mathbf{x}_1, x_i) \leq \overline{h_r(\mathbf{x}_b, \mathbf{x}_c)} \ . \tag{16}$$

Using (13)*, the inequality* (16) *can be split into the two inequalities*

$$\begin{aligned} h_q(\mathbf{x}_b, \mathbf{x}_c) &\leq h(\mathbf{x}_1, x_i) = x_i \oplus h'(\mathbf{x}_1) \\ h_q(\mathbf{x}_b, \mathbf{x}_c) &\leq (x_i \vee h'(\mathbf{x}_1))(\overline{x}_i \vee \overline{h'(\mathbf{x}_1)}) \ , \end{aligned} \tag{17}$$

and

$$\begin{aligned}
&\overline{h_r(\mathbf{x}_b,\mathbf{x}_c)} \geq h(\mathbf{x}_1,x_i) = x_i \oplus h'(\mathbf{x}_1)\\
&h_r(\mathbf{x}_b,\mathbf{x}_c) \leq \overline{h(\mathbf{x}_1,x_i)} = \overline{x}_i \oplus h'(\mathbf{x}_1)\\
&h_r(\mathbf{x}_b,\mathbf{x}_c) \leq (\overline{x}_i \vee h'(\mathbf{x}_1))(x_i \vee \overline{h'(\mathbf{x}_1)}) \; . \qquad (18)
\end{aligned}$$

The right-hand-side functions of (17) *and* (18) *are equal to or larger than the mark functions* $h_q(\mathbf{x}_b,\mathbf{x}_c)$ *and* $h_r(\mathbf{x}_b,\mathbf{x}_c)$*. The mark function* $h_r^{\partial x_i}(\mathbf{x}_1)$ *of Condition* (14) *is defined by:*

$$h_r^{\partial x_i}(\mathbf{x}_1) = \min_{x_i} h_q(x_i,\mathbf{x}_1) \vee \min_{x_i} h_r(x_i,\mathbf{x}_1) \; . \qquad (19)$$

Now, we substitute the function $h(\mathbf{x}_1,x_i)$ *of* (17) *for* $h_q(\mathbf{x}_b,\mathbf{x}_c)$ *and the function* $\overline{h(\mathbf{x}_1,x_i)}$ *of* (18) *for* $h_r(\mathbf{x}_b,\mathbf{x}_c)$ *into* (19)*. These functions are equal to or larger than the replaced functions. So we get:*

$$\begin{aligned}
h_r^{\partial x_i}(\mathbf{x}_1) = &\min_{x_i}\Big((x_i \vee h'(\mathbf{x}_1)) \wedge (\overline{x}_i \vee \overline{h'(\mathbf{x}_1)})\Big) \vee\\
&\min_{x_i}\Big((\overline{x}_i \vee h'(\mathbf{x}_1)) \wedge (x_i \vee \overline{h'(\mathbf{x}_1)})\Big)\\
= &\min_{x_i}(x_i \vee h'(\mathbf{x}_1)) \wedge \min_{x_i}(\overline{x}_i \vee \overline{h'(\mathbf{x}_1)}) \vee\\
&\min_{x_i}(\overline{x}_i \vee h'(\mathbf{x}_1)) \wedge \min_{x_i}(x_i \vee \overline{h'(\mathbf{x}_1)})\\
= &\left(h'(\mathbf{x}_1) \vee \min_{x_i}(x_i)\right) \wedge \left(\overline{h'(\mathbf{x}_1)} \vee \min_{x_i}(\overline{x}_i)\right) \vee\\
&\left(h'(\mathbf{x}_1) \vee \min_{x_i}(\overline{x}_i)\right) \wedge \left(\overline{h'(\mathbf{x}_1)} \vee \min_{x_i}(x_i)\right)\\
= &\left(h'(\mathbf{x}_1) \vee 0\right) \wedge \left(\overline{h'(\mathbf{x}_1)} \vee 0\right) \vee \left(h'(\mathbf{x}_1) \vee 0\right) \wedge \left(\overline{h'(\mathbf{x}_1)} \vee 0\right)\\
= &\; h'(\mathbf{x}_1) \wedge \overline{h'(\mathbf{x}_1)} \vee h'(\mathbf{x}_1) \wedge \overline{h'(\mathbf{x}_1)}\\
= &\; 0 \vee 0\\
= &\; 0 \; .
\end{aligned}$$

Hence, Condition (14) *is not only satisfied for the two mark functions* $h_q(\mathbf{x}_b,\mathbf{x}_c)$ *and* $h_r(\mathbf{x}_b,\mathbf{x}_c)$*, but also for the function* (13) *which is linear with regard to* x_i *and belongs to the evaluated lattice. That shows the implication* (13) $\Rightarrow$ (14) *and completes Theorem 1.*

Consequence 1 *An XOR-bi-decomposition is compact, if the set of variables* $\mathbf{x}_b$ *is as large as possible (verified by Condition* (3)*), and within an iterative procedure*

all variables x_i of the initial set $\mathbf{x}_c$ that satisfy Condition (14) *are used to transform $g(\mathbf{x}_{a0},\mathbf{x}_{c0},x_i)$ to $g'(\mathbf{x}_a,\mathbf{x}_{c0})$ by*

$$g'(\mathbf{x}_a,\mathbf{x}_{c0}) = x_i \oplus g(\mathbf{x}_{a0},\mathbf{x}_{c0},x_i) \tag{20}$$

and the associated new lattice $L\left\langle h'_q(\mathbf{x}_b,\mathbf{x}_{c0}),h'_r(\mathbf{x}_b,\mathbf{x}_{c0})\right\rangle$ is adjusted by

$$h'_q(\mathbf{x}_b,\mathbf{x}_{c0}) = \max_{x_i}\left(\overline{x}_i\, h_q(\mathbf{x}_b,\mathbf{x}_{c0},x_i) \vee x_i\, h_r(\mathbf{x}_b,\mathbf{x}_{c0},x_i)\right)\ , \tag{21}$$

$$h'_r(\mathbf{x}_b,\mathbf{x}_{c0}) = \max_{x_i}\left(\overline{x}_i\, h_r(\mathbf{x}_b,\mathbf{x}_{c0},x_i) \vee x_i\, h_q(\mathbf{x}_b,\mathbf{x}_{c0},x_i)\right)\ . \tag{22}$$

A precondition for a compact XOR-bi-decomposition is the existence of two variables x_a and x_b for which the given lattice $L\left\langle f_q(\mathbf{x}),f_r(\mathbf{x})\right\rangle$ contains at least one function which has a strong XOR-bi-decomposition with regard to these variables. Algorithm 1 analyzes whether there is an XOR-bi-decomposition for the given lattice with regard to one pair of variables x_a and x_b. Algorithm 1 determines these variables if they exist.

Algorithm 1 Initial XOR-bi-decomposition of the lattice $L\left\langle f_q(\mathbf{x}),f_r(\mathbf{x})\right\rangle$ with regard to the variables x_a and x_b

Require: TVLs of $f_q(\mathbf{x})$and $f_r(\mathbf{x})$ in ODA-form
Ensure: Boolean variable *hasXORbd*: it is $true$ if the given lattice contains at least one XOR-bi-decomposable function and $false$ otherwise
Ensure: set of variables (SV) of x_a and x_b: variables for which the lattice contains at least one function with a strong XOR-bi-decomposition

1: $all\_var \leftarrow \text{SV\_UNI}(f_q,f_r)$
2: $hasXORbd \leftarrow false$
3: $x_a \leftarrow \emptyset$
4: **while** $\overline{hasXORbd} \wedge \text{SV\_NEXT}(all\_var,x_a,x_a)$ **do**
5: $\quad x_b \leftarrow x_a$
6: $\quad f_q^{\partial x_a} \leftarrow \text{ISC}(\text{MAXK}(f_q,x_a),\text{MAXK}(f_r,x_a))$
7: $\quad f_r^{\partial x_a} \leftarrow \text{UNI}(\text{MINK}(f_q,x_a),\text{MINK}(f_r,x_a))$
8: $\quad$ **while** $\overline{hasXORbd} \wedge \text{SV\_NEXT}(all\_var,x_b,x_b)$ **do**
9: $\quad\quad$ **if** $\text{TE\_ISC}(\text{MAXK}(f_q^{\partial x_a},x_b),f_r^{\partial x_a})$ **then**
10: $\quad\quad\quad hasXORbd \leftarrow true$
11: $\quad\quad$ **end if**
12: $\quad$ **end while**
13: **end while**
14: **return** $(hasXORbd,x_a,x_b)$

Algorithm 1 uses two nested while-loops for the selection of the variables x_a and x_b. The basic set of all variables is prepared in line 1 using the XBOOLE-function SV_UNI (set of variables - union). The sequential selection of the variables x_a and x_b is realized by two XBOOLE-functions SV_NEXT (set of variables - next variable) that control these while-loops. The variable *hasXORbd* is used to indicate the Boolean result whether the lattice contains at least one function with a strong XOR-bi-decomposition. This variable is also used to terminate both while-loops if a strong XOR-bi-decomposition is detected.

XBOOLE-functions ISC (intersection), UNI (union), MAXK (k-fold maximum), and MINK (k-fold minimum) calculate in lines 6 and 7 the mark functions of the derivative of the given lattice with regard to the single variables x_a. XBOOLE-function TE_ISC (test empty - intersection) checks in line 9 Condition (3) for the strong XOR-bi-decomposition with regard to the actually selected variables x_a and x_b. In the case that this condition is satisfied, the control variable *hasXORbd* is changed to the value *true* in line 10.

Algorithm 2 extends a found initial XOR-bi-decomposition to a compact one. Initial steps determine all variables (line 1), the basic sets of commonly used variables $\mathbf{x}_c$ (line 2) and dedicated variables $\mathbf{x}_b$ (line 3), and the precondition (line 4) for the selection of variables x_b by means of the XBOOLE-function SV_NEXT in line 8.

The mark functions of the derivative of the given lattice with regard to the single variables x_a are needed in Condition (3) to decide about the possibility to extend the set $\mathbf{x}_b$; they are calculated in lines 5 and 6 based on (1) and (2). The help-function h_0 stores the intermediate result of the k-fold maximum with regard to the already known variables of the set $\mathbf{x}_b$ (line 7).

The while-loop in lines 8 to 13 extends the set $\mathbf{x}_b$ to the maximal possible number of variables of a strong XOR-bi-decomposition for the given lattice. Condition (3) is verified in line 9 for the temporally extended set $\mathbf{x}_b$. If this condition is satisfied for the set of variables $\mathbf{x}_b \cup x_b$, the set of variables $\mathbf{x}_b$ is permanently extended in line 10 and the help-function h_0 is adjusted in line 11.

Knowing the maximal set of variables $\mathbf{x}_b$, basic versions of the wanted functions can be calculated:

- $g(x_a, \mathbf{x}_c)$ based on (4) in line 14;
- $h_q(x_b, \mathbf{x}_c)$ based on (5) in line 15; and
- $h_r(x_b, \mathbf{x}_c)$ based on (6) in line 16.

In a second while-loop (lines 20 to 28) the set of variables $\mathbf{x}_a$ is extended. Initial steps determine the new set of commonly used variables $\mathbf{x}_c$ (line 17), the basic set

Algorithm 2 Compact strong XOR-bi-decomposition of the lattice $L\langle f_q(\mathbf{x}), f_r(\mathbf{x})\rangle$

Require: TVLs of $f_q(\mathbf{x})$ and $f_r(\mathbf{x})$; initial SVs of x_a and x_b
Ensure: TVL of $g(\mathbf{x}_a, \mathbf{x}_c)$: decomposition function
Ensure: TVLs of $h_q(\mathbf{x}_b, \mathbf{x}_c)$ and $h_r(\mathbf{x}_b, \mathbf{x}_c)$: decomposition lattice
Ensure: SVs of $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$: disjoint sets of variables

1: $all\_var \leftarrow \text{SV\_UNI}(f_q, f_r)$
2: $\mathbf{x}_c \leftarrow \text{SV\_DIF}(\text{SV\_DIF}(all\_var, x_a), x_b))$
3: $\mathbf{x}_b \leftarrow x_b$
4: $x_b \leftarrow \emptyset$
5: $f_q^{\partial x_a} \leftarrow \text{ISC}(\text{MAXK}(f_q, x_a), \text{MAXK}(f_r, x_a))$
6: $f_r^{\partial x_a} \leftarrow \text{UNI}(\text{MINK}(f_q, x_a), \text{MINK}(f_r, x_a))$
7: $h_0 \leftarrow \text{MAXK}(f_q^{\partial x_a}, \mathbf{x}_b)$
8: **while** $\text{SV\_NEXT}(\mathbf{x}_c, x_b, x_b)$ **do**
9: **if** $\text{TE\_ISC}(\text{MAXK}(h_0, x_b), f_r^{\partial x_a})$ **then**
10: $\mathbf{x}_b \leftarrow \text{SV\_UNI}(\mathbf{x}_b, x_b)$
11: $h_0 \leftarrow \text{MAXK}(f_q^{\partial x_a}, \mathbf{x}_b)$
12: **end if**
13: **end while**
14: $g \leftarrow \text{ISC}(\text{SV\_GET}(x_a), h_0)$
15: $h_q \leftarrow \text{MAXK}(\text{UNI}(\text{ISC}(\overline{g}, f_q), \text{ISC}(g, f_r)), x_a)$
16: $h_r \leftarrow \text{MAXK}(\text{UNI}(\text{ISC}(\overline{g}, f_r), \text{ISC}(g, f_q)), x_a)$
17: $\mathbf{x}_c \leftarrow \text{SV\_DIF}(\text{SV\_DIF}(all\_var, x_a), \mathbf{x}_b))$
18: $\mathbf{x}_a \leftarrow x_a$
19: $x_i \leftarrow \emptyset$
20: **while** $\text{SV\_NEXT}(\mathbf{x}_c, x_i, x_i)$ **do**
21: **if** $\text{TE\_UNI}(\text{MINK}(h_q, x_i), \text{MINK}(h_r, x_i))$ **then**
22: $\mathbf{x}_a \leftarrow \text{SV\_UNI}(\mathbf{x}_a, x_i)$
23: $x_i \leftarrow \text{SV\_GET}(x_i)$
24: $g \leftarrow \text{SYD}(x_i, g)$
25: $h_q \leftarrow \text{MAXK}(\text{UNI}(\text{ISC}(\overline{x}_i, h_q), \text{ISC}(x_i, h_r)), x_i)$
26: $h_r \leftarrow \text{MAXK}(\text{UNI}(\text{ISC}(\overline{x}_i, h_r), \text{ISC}(x_i, h_q)), x_i)$
27: **end if**
28: **end while**
29: $\mathbf{x}_c \leftarrow \text{SV\_DIF}(\mathbf{x}_c, \mathbf{x}_a)$
30: **return** $(g, h_q, h_r, \mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c)$

of variables $\mathbf{x}_a$ (line 18), and the selection variable x_i (line 19) needed to evaluate the possibility of the extension of $\mathbf{x}_a$.

Condition (14) of Theorem 1 is verified in line 21 using the formula (2) to determine the OFF-set of the derivative of a lattice with regard to the single variable x_i. If this condition is satisfied:

- the set of variables $\mathbf{x}_a$ is extended by x_i in line 22 using the XBOOLE-function SV_UNI (set of variables - union);
- x_i is transformed in line 23 from a set of variables into the TVL representing $x_1 = 1$ using the XBOOLE-function SV_GET (set of variables - get);
- the new function g is calculated in line 24 based on (20) using the XBOOLE-function SYD (symmetric difference);
- the new ON-set function $h_q(\mathbf{x})$ is calculated in line 25 based on (21); and
- the new OFF-set function $h_r(\mathbf{x})$ is calculated in line 26 based on (22).

The restriction of the set of commonly used variables $\mathbf{x}_c$ is realized in line 29 outside of the loop, because the unchanged basic set $\mathbf{x}_c$ is needed in the XBOOLE-function SV_NEXT in line 20 to select the next variable x_i. The complement operations in lines 15, 16, 24, and 25 are realized using the XBOOLE-function CPL.

5 EXAMPLE

5.1 The Chosen Lattice and Conditions for the Synthesis

(a)

x_4	x_5	$L\langle f_q(\mathbf{x}), f_r(\mathbf{x})\rangle$							
0	0	0	1	0	1	0	1	0	1
0	1	1	1	0	0	1	1	Φ	0
1	1	0	1	0	1	1	0	0	1
1	0	Φ	1	0	0	0	Φ	0	0
		0	1	1	0	0	1	1	0 x_3
		0	0	1	1	1	1	0	0 x_2
		0	0	0	0	1	1	1	1 x_1

(b)

x_4	x_5	$y = f(\mathbf{x})$							
0	0	0	1	0	1	0	1	0	1
0	1	1	1	0	0	1	1	0	0
1	1	0	1	0	1	1	0	0	1
1	0	1	1	0	0	0	0	0	0
		0	1	1	0	0	1	1	0 x_3
		0	0	1	1	1	1	0	0 x_2
		0	0	0	0	1	1	1	1 x_1

Fig. 1. Karnaugh-maps of (a) the given lattice, and (b) chosen function of both bi-decompositions.

Figure 1 (a) shows the Karnaugh-map of a lattice of eight Boolean functions. The simplest multi-level circuit structure for one of these functions must be found using AND-, OR-, and XOR-gates of two inputs where these inputs arbitrary can

be negated. The gates can be reused to simplify the circuit. As basis for comparison serves a minimal disjunctive form, calculated by means of the well known Quine McCluskey algorithm. The synthesis of the given lattice of functions by bi-decompositions has been realized using both the know non-compact XOR-bi-decomposition and the new compact XOR-bi-decomposition. Using conditions given in [3, 4, 11] it can be verified that this lattice does not contain any function which has a strong bi-decomposition with regard to any dedicated sets of variables $\mathbf{x}_a$ and $\mathbf{x}_b$ for an AND- or an OR-gate. Figure 1 (b) shows the function chosen by both the known and the new bi-decomposition approach. Two don't-cares are assigned to 0 and the other to 1. The simplest minimal disjunctive form realizes the function $f_q(\mathbf{x})$ of the lattice where all don't-cares are assigned to 0.

5.2 Synthesis by Covering Using a Minimal Disjunctive Form

The execution of the Quine McCluskey algorithm results in two minimal disjunctive forms of the same complexity. Both of them realize the ON-set function $f_q(\mathbf{x})$ and require the same number of gates and levels in a circuit. The chosen minimal disjunctive form is:

$$f_q(\mathbf{x}) = (\bar{x}_1\bar{x}_2)x_3 \vee (x_1x_2)(\bar{x}_4x_5) \vee (\bar{x}_1\bar{x}_2)(\bar{x}_4x_5) \vee (x_2\bar{x}_3)(x_4x_5) \vee (x_1x_2)(x_3\bar{x}_4) \vee (x_1\bar{x}_3)(x_4x_5) \vee (\bar{x}_1(x_2\bar{x}_3))(\bar{x}_4\bar{x}_5) \vee (\bar{x}_2(x_1\bar{x}_3))(\bar{x}_4\bar{x}_5) \; . \quad (23)$$

The parentheses in the conjunctions in (23) emphasize the chosen two-input AND-gates. Figure 2 shows the associated circuit structure in which as much as possible AND-gates are reused.

The disjunction of eight conjunctions is realized by a tree of seven OR-gates. Seven AND-gates could be reused to build another conjunction. In total there are 18 AND-gates. The complete circuit consists of 25 two-input gates on six levels.

5.3 Synthesis Using the Known Non-Compact XOR-Bi-Decomposition

Using Condition (3) it was found that the lattice of Figure 1 (a) contains at least one function that is XOR-bi-decomposable with regard to the single variable $x_a = x_1$ and the dedicated set $\mathbf{x}_b = (x_3, x_5)$. Hence, the set of commonly used variables $\mathbf{x}_c = (x_2, x_4)$.

The decomposition function $g(x_a, \mathbf{x}_c)$ of an XOR-bi-decomposition is uniquely specified by (4), and we get

$$g_1(x_1, x_2, x_4) = x_1 \wedge \overline{(x_2 \wedge x_4)} \; . \quad (24)$$

It can directly be seen that there is a strong AND-bi-decomposition of $g_1(x_1, x_2, x_4)$ into $g_2 = x_1$ and $h_2 = \overline{(x_2 \wedge x_4)}$. No further decomposition is needed for these functions.

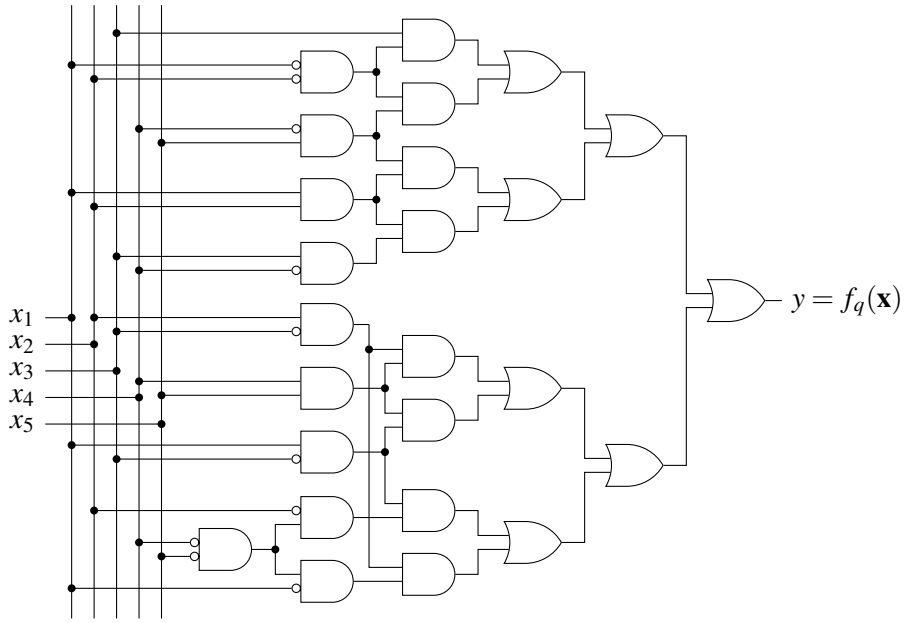

Fig. 2. Circuit structure synthesized by Quine-McCluskey and reused two-input gates.

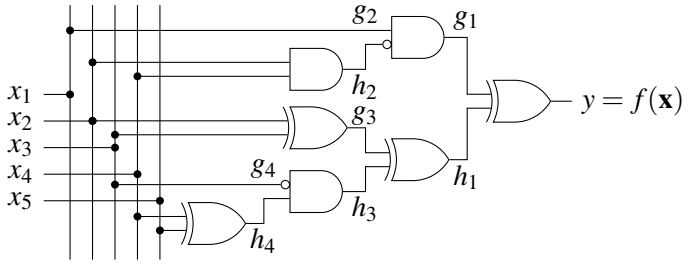

Fig. 3. Circuit structure synthesized using the old non-compact XOR-bi-decomposition.

The lattice of the decomposition function h_1 can be calculated by (5) and (6) and contains in this example the single function

$$h_1(x_2,x_3,x_4,x_5) = (\overline{x}_3 \wedge (x_4 \oplus x_5)) \oplus (x_2 \oplus x_3)\ . \tag{25}$$

By means of Condition (3) it can be verified that an XOR-bi-decomposition of $L\langle h_{1q}, h_{1r}\rangle$ with regard to $x_a = x_2$ and $\mathbf{x}_b = (x_4, x_5)$ exists. Hence, only the variable x_3 belongs to the set of commonly used variables $\mathbf{x}_c$.

The decomposition function $g(x_a, \mathbf{x}_c)$ of this second XOR-bi-decomposition

was again calculated by (4):

$$g_3(x_2,x_3) = x_2 \oplus x_3 \ .$$

The lattice $L\left\langle h_{3q}, h_{3r}\right\rangle$ contains only the single function

$$h_3(x_3,x_4,x_5) = \overline{x}_3 \wedge (x_4 \oplus x_5)$$

for which a strong AND-bi-decomposition into $g_4 = \overline{x}_3$ and $h_4 = (x_4 \oplus x_5)$ exists. No further decomposition is needed for these functions. Figure 3 shows the synthesized circuit consisting of seven gates on four levels.

5.4 Optimized Synthesis Using the New Compact XOR-Bi-Decomposition

For direct comparison we demonstrated the approach of the utilization of a linearly separable variable to get a compact XOR-bi-decomposition using the same lattice (shown in Figure 1 (a)) as before.

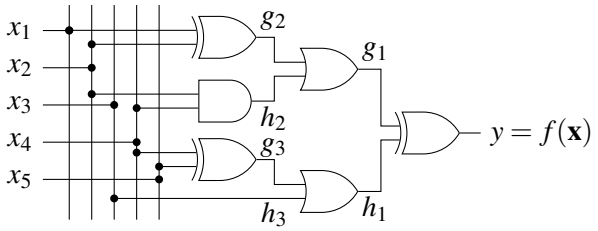

Fig. 4. Circuit structure synthesized using the new compact XOR-bi-decomposition.

Using Condition (3) in Algorithm 1 the initial XOR-bi-decomposition with regard to the single variables $x_a = x_1$ and $x_b = x_3$ is found. In the first part of Algorithm 2 (lines 1 to 13) the dedicated set $\mathbf{x}_b$ could be extended to (x_3,x_5) due to the check of variables x_2, x_4, and x_5 within line 9 embedded in the loop of lines 8 to 13. Hence, the basic set of commonly used variables is $\mathbf{x}_c = (x_2,x_4)$ which is determined in line 17.

Algorithm 2 finds by Condition (14) in line 21 in the while-loop in lines 20 to 28 that the so fare detected lattice of h_1 contains a function that is linear with regard to x_2. Hence, x_2 is included into the set $\mathbf{x}_a$ in line 22, the new function g_1 is calculated in lines 23 and 24 using the basic function g_1' (24):

$$\begin{aligned} g_1(x_1,x_2,x_4) &= x_2 \oplus g_1'(x_1,x_2,x_4) \\ &= x_2 \oplus \left(x_1 \wedge \overline{(x_2 \wedge x_4)}\right) \\ &= (x_1 \oplus x_2) \vee (x_2 \wedge x_4) \ . \end{aligned} \tag{26}$$

Using (21) and (22) the new lattice $L\langle h_{1q}, h_{1r}\rangle$ is calculated in lines 25 and 26 of Algorithm 2. Due to the special case of the completely specified function h_1' (25) the result is also a completely specified function that is calculated by (21):

$$\begin{aligned} h_1(x_3,x_4,x_5) &= \max_{x_2}\left(x_2 \oplus h_1'(x_2,x_3,x_4,x_5))\right) \\ &= \max_{x_2}\left(x_2 \oplus (\bar{x}_3 \wedge (x_4 \oplus x_5)) \oplus (x_2 \oplus x_3)\right) \\ &= (\bar{x}_3 \wedge (x_4 \oplus x_5)) \oplus x_3 \\ &= (x_4 \oplus x_5) \vee x_3 \ . \end{aligned} \tag{27}$$

Hence, $h_1(x_3,x_4,x_5)$ depends only on three variables, and the comparison with $g_1(x_1,x_2,x_4)$ confirms that only the variable x_4 is shared. In this way the single variable of the dedicated set $\mathbf{x}_a$ is implicitly extended to $\mathbf{x}_a = (x_1,x_2)$. Algorithm 2 explicitly realizes this extension in line 22 using the XBOOLE operation SV_UNI (set of variables - union).

The expressions (26) and (27) show that there are OR-bi-decompositions for both decomposition functions g_1 and h_1. Figure 4 shows that the circuit structure, realized be means of the new method, only needs six gates on three levels.

5.5 Comparison of the Synthesis Results

Table 1 summarizes the results of the synthesis of the given lattice of Boolean functions realized by:

- the covering method using the Quine McCluskey approach to get a minimal disjunctive form which has been split into two-input gates that are reused as much as possible;
- the bi-decomposition method where the known XOR-bi-decomposition of a lattice is restricted to the assignment of a single variable to the dedicated set $\mathbf{x}_a$;
- the bi-decomposition method using the new XOR-bi-decomposition for a lattice that is able to realize a compact XOR-bi-decomposition.

Both the needed area and the power consumption are estimated by the number of gates. The benefit of the bi-decomposition in comparison to the covering method is evident; the number of gates could be reduced, despite the seven reused gates in the covering approach, from 25 to seven in case of the known bi-decomposition and even to six when the new compact XOR-bi-decomposition is used. This is a reduction to 24% of the needed area as well as the power consumption of the new

Table 1. Comparison of needed area, power consumption, and maximal delay

effect to	used count	covering method	used XOR-bi-decomposition		ratios	
			known	new compact	$\frac{\text{new compact}}{\text{covering}}$	$\frac{\text{new compact}}{\text{known}}$
area	number of gates	25	7	6	24.0 %	85.7 %
power	number of gates	25	7	6	24.0 %	85.7 %
delay	number of gates in the longest path	6	4	3	50.0 %	75.0 %

compact XOR-bi-decomposition in comparison to the covering method or to 85.7% regarding the so far used non-compact XOR-bi-decomposition.

The maximal delay of the synthesized circuit can be estimated by the number of gates in the longest path that is equal to the number of gate levels. The bi-decomposition outperforms the covering method also regarding the maximal delay. The new compact XOR-bi-decomposition was able to reduce the maximal delay to one half in comparison to the covering method or 75% according to the known non-compact XOR-bi-decomposition.

6 CONCLUSIONS

Lattices of Boolean functions provide the possibility to choose the function for which the circuit needs a small area, a low power consumption, and has a short delay time. The bi-decomposition is a very powerful method to synthesize circuits that improve these parameters in comparison to covering methods. The theory to find compact strong bi-decompositions was so far only known for AND- and OR-gates. However, strong XOR-bi-decompositions were restricted to a single variable in the dedicated set $\mathbf{x}_a$.

The results of this paper close this gap of a missing compact XOR-bi-decomposition for lattices of Boolean functions. It provides both the needed new theory and their application in Algorithms using XBOOLE [19, 20] for the calculation of compact XOR-bi-decompositions for lattices of Boolean functions.

In a very simple example the gate count (needed area, power consumption) could be reduced to 24 percent in comparison to an exact covering method and to 85 percent regarding the known bi-decomposition. For the same example the length of the longest path (maximal delay) could be reduced to one half in comparison to an exact covering method and to 75 percent according to the known bi-decomposition.

REFERENCES

[1] D. Bochmann, F. Dresig, and B. Steinbach. “A New Decomposition Method for Multilevel Circuit Design”. In: *Proceedings of the Conference on European Design Automation*. EDAC ’91. Amsterdam, The Netherlands: IEEE Computer Society, 1991, pp. 374–377.

[2] T. Le. “Testbarkeit kombinatorischer Schaltungen - Theorie und Entwurf”. written in German, English title: Testability of Combinational Circuits - Theory and Design. PhD thesis. TU Karl-Marx-Stadt, Germany, 1989.

[3] C. Posthoff and B. Steinbach. *Logic Functions and Equations – Binary Models for Computer Science*. Dordrecht, The Netherlands: Springer, 2004.

[4] B. Steinbach and C. Posthoff. *Boolean Differential Calculus*. Synthesis Lecturers on Digital Circuits and Systems 52. San Rafael, CA, USA: Morgan & Claypool, 2017.

[5] A. Mishchenko, B. Steinbach, and M. Perkowski. “An Algorithm for Bi-decomposition of Logic Functions”. In: *Proceedings of the 38th Annual Design Automation Conference*. DAC ’01. Las Vegas, Nevada, USA: ACM, 2001, pp. 103–108.

[6] B. Steinbach. “Vectorial Bi-Decompositions of Logic Functions”. In: *Proceedings of the Reed-Muller Workshop 2015*. RM 4. Waterloo, Canada, 2015.

[7] B. Steinbach and C. Posthoff. “Vectorial Bi-Decompositions for Lattices of Boolean Functions”. In: *Proceedings of the 12th International Workshops on Boolean Problems*. IWSBP. Freiberg, Germany: Freiberg University of Mining and Technology, 2016, pp. 93–104.

[8] A. Thayse. “Boolean Differential Calculus”. In: *Philips Research Reports* 26 (1971). R 764, pp. 229–246.

[9] M. Davio and A. Thayse. “Boolean Differential Calculus and its Application to Switching Theory”. In: *IEEE Transactions on Computes* 22.4 (1973), pp. 409–420.

[10] B. Steinbach and C. Posthoff. *Logic Functions and Equations - Examples and Exercises*. Springer Science + Business Media B.V., 2009.

[11] B. Steinbach and C. Posthoff. “Boolean Differential Calculus - Theory and Applications”. In: *Journal of Computational and Theoretical Nanoscience* 7.6 (2010), pp. 933–981.

[12] B. Steinbach and C. Posthoff. “Boolean Differential Calculus”. In: *Progress in Applications of Boolean Functions*. Synthesis Lecturers on Digital Circuits and Systems 26. San Rafael, CA, USA: Morgan & Claypool, 2010, pp. 55–78.

[13] T. Sasao and J. Butler. “On Bi-Decompositions of Logic Functions”. In: *6th International Workshop on Logic & Synthesis*. IWLS. Granlibakken Resort - Tahoe City, CA, USA, 1997, pp. 1–6.

[14] M. Choudhury and K. Mohanram. “Bi-Decomposition of Large Boolean Functions Using Blocking Edge Graphs”. In: *2010 IEEE/ACM International Conference on Computer-Aided Design*. ICCAD. 2010, pp. 586–591.

[15] D. Cheng and X. Xu. “Bi-Decomposition of Logical Mappings via Semi-Tensor Product of Matrices”. In: *Automatica* 49.7 (2013), pp. 51–76.

[16] B. Steinbach. “Generalized Lattices of Boolean Functions Utilized for Derivative Operations”. In: *Materiały konferencyjne KNWS’13*. KNWS ’13. Łagów, Poland, 2013, pp. 1–17.

[17] B. Steinbach. “Derivative Operations for Lattices of Boolean Functions”. In: *Proceedings of the Reed-Muller Workshop 2013*. RM ’13. Toyama, Japan, 2013, pp. 110–119.

[18] B. Steinbach and A. Wereszczynski. “Synthesis of Multi-Level Circuits Using EXOR-Gates”. In: *IFIP WG 10.5 - Workshop on Applications of the Reed-Muller Expansion in Circuit Design*. Chiba - Makuhari, Japan, 1995, pp. 161–168.

[19] B. Steinbach. “XBOOLE - A Toolbox for Modelling, Simulation, and Analysis of Large Digital Systems”. In: *Systems Analysis and Modelling Simulation* 9.4 (1992), pp. 297–312.

[20] B. Steinbach and M. Werner. “XBOOLE-CUDA - Fast Calculations of Large Boolean Problems on the GPU”. In: *Problems and New Solutions in the Boolean Domain*. Ed. by B. Steinbach. Newcastle upon Tyne, UK: Cambridge Scholars Publishing, 2016, pp. 117–149.