Automation and Remote Control, Vol. 65, No. 6, 2004, pp. 893–912. Translated from Avtomatika i Telemekhanika, No. 6, 2004, pp. 61–83.

ARITHMETIC LOGIC

Malyugin's Theorems: A New Concept in Logical Control, VLSI Design, and Data Structures for New Technologies[1]

V. P. Shmerko

University of Calgary, Calgary, Canada
Strzelin Technical University, Strzelin, Poland

Received December 16, 2003

Abstract—Theoretical results due to V. Malyugin on linearization of arithmetical models of logical functions are interpreted and refined from the viewpoint of modern techniques of logical design of integral circuits and nearest neighbor technologies.

1. INTRODUCTION

Computer-aided design of integral circuits (CAD IC) consists in transforming the description of a circuit at the algorithmic or behaviorial level to a description at the physical level satisfying technological conditions. This process is subdivided into *system*, *behavioral*, *logic* and *layout* levels. This paper is concerned with the *logic level*, in which the behavior of a circuit is described by logical functions. The effectiveness of algorithms used in this level primarily depends on data structures (AND/OR forms, cubes, vector representations, etc.). Problems of large dimension are effectively solved with decision diagrams (DD) [1, 2]. Arithmetical logic is the extension of the Boolean algebra through arithmetical operations permitting the representation of logical functions by integer functions (Boolean-to-integer functions). In many cases, this is helpful in manipulating and transforming data of CAD systems [3, 4].

Beginning from the pioneering works of Aiken on the application of arithmetical logic in designing the computers MARK 3 and MARK 4 (1952) [5], many papers have developed *word-level* structures, i.e., representation of data in the form of specially grouped words, to obtain compact data representation and thereby increase the dimension of problems, for example, by the so-called *word-level DD*. Another approach is developed in [6]. However, it is not clear which of the methods of formation of words from Boolean functions is preferable. The properties of arithmetical logic in this respect are rather attractive, for example, is it not possible to group data such that they are described by a linear arithmetical polynomial (LAR). In certain well-know publications, this problem is solved in a simplified formulation by *partly linearizing the word-level model* [1, 3, 4, 7]. Fundamental results due to Malyugin [8, 9] pave the way for solving this problem.

This paper is devoted to *linearization of arithmetical word-level models*—an integral part of the Malyugin approach. Our aim is to interpret his approach to linearization of arithmetical models (polynomials) and refine it from the standpoint of new requirements and capabilities of modern CAD IC systems. In particular, we describe a classification for word-level models and show the importance of the cortege technique for linearization of arithmetical models. Jointly with linearization conditions and matrix transformation technique [10, 11], the arithmetical model of a circuit (combination circuit or a circuit with memory on binary and multilevel gates) is shown to be linearizable [12]. Then, the linear model is directly mapped into a linear DD (LDD) having certain useful properties. For the first time we show that *any circuit is representable by a system of LDD.*

[1] This work was supported in part by the NATO Collaborative Linkage Grant PST/CLG 979071.

0005-1179/04/6506-0893

Finally, using the properties of linear models, we solve the *problem of large weight coefficients* characteristic of arithmetical forms. It is a serious hurdle in practical application of arithmetical logic. Experiments with industrial circuits have shown that our algorithm is effective for solving large-dimensional problems (in experiments, coefficients took values up to 2^{100}); none of the known methods can solve such problems. Latest results in arithmetical logic, including linearization of models, are reviewed in [13].

Arithmetic transformation methods are discussed in Sections 3–5. Linearization of arithmetic polynomials is described in Section 6. Section 7 is concerned with the computational aspects of linear transforms based on LAR. Section 8 is devoted to manipulations with large values of coefficients of LAR. Experimental results are given in Section 9. The application of arithmetic logic in prospective technologies is examined in Section 10.

2. PRELIMINARIES

2.1. Word-level Forms

A Boolean function of n variables is a mapping $\{0,1\}^n \to \{0,1\}$. The word-level form in arithmetic logic is an *integer function* P with a mapping $\{0,1\}^n \to \{0,1,\ldots,p-1\}$, where $p>2$. In general, a system of r Boolean functions of n variables (r-output Boolean function) can be represented by an integer function D with the mapping $\{0,1\}^n \to \{0,1,\ldots,2^r-1\}$. Word-level representation of a system of Boolean functions is defined to be an *ordering of integer functions P_j such that the functions form a word*

$$D = 2^{r-1}P_r + \ldots + 2^1P_2 + 2^0P_1. \tag{1}$$

The outputs $f_1, f_2, \ldots, f_r$ of a Boolean function f are uniquely estimated from expression (1) through transformation of P_j into f_j. This form is the *weighted sum of r arithmetic polynomials P_j*, $j=1,\ldots,r$. A particular case of this expression is the *linear arithmetic polynomial* (LAR), which contains not more than $(n+1)$ nonzero coefficients. Transformation of an arithmetic polynomial (1) to a LAR is the *solution of the boundary-value problem of arithmetic logic*. In this paper, we focus our attention on methods of formation of LAR and their properties.

In the general case, a word-level model is described by a *sequence of words*. A word contains information on a function and components of a system (circuit). The simplest word-level model contains only Boolean constants. For example, the word 101 represents a binary word. In CAD IC, we use different types of word-level models, for instance, sum of weighted Boolean expressions or arithmetic polynomials (1). Word-level structures are manipulated and transformed by a special technique based on decision diagrams, known as the WDD [2]. The initial word-level description may contain arithmetical expressions or truth vectors of Boolean functions. Malyugin refers to such data structures as *corteges* and has developed a technique for manipulating over them [8].

2.2. Linearization Problem

Arithmetic logic is an *extension of Boolean algebra*. An arithmetical expression is formed via replacement of Boolean operations by arithmetical operations over integer data according to certain rules. Algebra of corteges is an *extension of arithmetical logic*.[2]

[2] *An extension* of a field K is obtained by including new elements $r_1, r_2, \ldots, r_n$ such that $F = K(r_1, r_2, \ldots, r_n)$, where $r_1, r_2, \ldots, r_n \in F$. The resultant field F is called the extension of the field K if all group operations and field properties are preserved.

Definition 1. The LAR of a Boolean function f of n variables $x_1, \ldots, x_n$ is

$$D = d_0 + \sum_{i=1}^{n} d_i x_i = d_0 + d_1 x_1 + \ldots + d_n x_n, \tag{2}$$

where d_i $(i = 1, 2, \ldots, n)$ is the ith integer coefficient.

According to expression (2), an r-output Boolean function can be represented by a LAR if all polynomials P_j in (1) are LARs (trivial case) or (b) the weighted sum of (1) contains LARs and nonlinear polynomials P_j (the general case).

Linearization means transformation of a nonlinear arithmetic polynomial (1) into a linear arithmetic polynomial (2) with not more than $(n+1)$ nonzero coefficients. Probably, a similar linearization problem was first solved in [16], though general interest in this topic had emerged as early as 1952. (see [17]). A solution in the spectral domain is given in [18, 19].

Example 1. The outputs $f_1 = x_1 \oplus x_2$ and $f_2 = x_1 x_2$ of a half-adder are nonlinear arithmetic polynomials $P_1 = x_1 + x_2 - 2x_1 x_2$ and $P_2 = x_1 x_2$, respectively. This is the general case of design of LARs (2). According to (1), $D = 2^1 P_2 + 2^0 P_1 = 2^1 x_1 x_2 + 2^0 (x_1 + x_2 - 2x_1 x_2) = x_1 + x_2$ is a LAR. Interchanging the numbers of P_1 and P_2, we obtain a *nonlinear* polynomial $D = 2x_1 + 2x_2 - 3x_1 x_2$.

In this example, identical nonlinear terms $2x_1 x_2$ have been eliminated as a result of their unlike signs by a suitable choice of order of numeration for the outputs of the Boolean function. Consequently, linearization of expression (1) is *sensitive* to the order of numeration of outputs of the Boolean function in the word-level representation. If this order is suitably chosen, then the LAR uniquely represents a function, and this provides a method for *estimating* the initial function.

The linearization problem for data structures can also be formulated differently: find a linear transformation L for the variables $x_1, x_2, \ldots, x_n$ of a Boolean function f such that $f = L\{\widetilde{f}\}$. If the Boolean function $\widetilde{f}$ is simpler to compute than the function f and the transformation L does not require sophisticated computation resources, then this approach is useful in designing and analyzing logical circuits [7, 20, 21].

2.3. Topic of Study

The state-of-the-art of our problem shows that there are two main hurdles in the way of practical application of arithmetical logic to logical design and control, viz.:

(*i*) *Absence of satisfactory methods for linearization of arithmetical models.* Although the results due to Malyugin give a "key" to many real problems, his linearization approach needs refinement from the viewpoint of modern CAD IC technology.

(*ii*) *Impossibility of utilization of the existing methods for solving large-dimensional problems.* Description of data structures by arithmetical polynomials and manipulations over them requires huge weight coefficients even for small circuits. If this hurdle is surmounted, vistas open for the utilization of the word-level technique, in particular, WDD technology.

Motivated by these facts, we shall focus our attention on *linear models of arithmetical logic for multilevel circuits and for transforming them to an LDD set*:

$$\text{Function (circuit)} \iff \text{LAR} \iff \text{LDD} \iff \text{Realization}.$$

Thus we can attain the main goal: application of Malyugin's results in CAD IC, their interpretation in terms of modern word-level technology, and solution of certain related problems, such as transmission of data via LDD networks and manipulations with large weight coefficients.

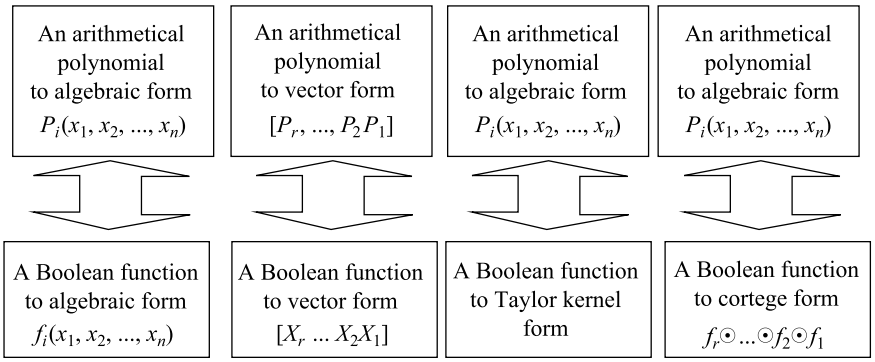

Fig. 1. Four methods of transformation of a Boolean function to an arithmetical polynomial (left to right): algebraic, matrix, Taylor expansion, via algebra of corteges (Malyugin's method).

To determine the place and role of Malyugin's linearization methods in modern word-level technology, we must classify methods of realization of *arithmetical transformations*, by which we mean the relationship between Boolean functions and their representation by arithmetical polynomials.[3]

Direct arithmetical transformation is used for solving the following problem: given a Boolean function (a many-output function in the general case), find its corresponding arithmetical polynomial. *Inverse* arithmetical transformation is used for estimating the Boolean function from a given polynomial. Consequently, arithmetical transformation is defined by a pair of relations for solving direct and inverse problems.

In this paper, we use four methods for manipulations over arithmetical expressions: algebraic, matrix, Taylor expansion, and algebra of corteges based on the Malyugin method (Fig. 1).[4]

Algebraic manipulations based on the linearity of arithmetical transformations. They feature clarity and simplicity, and helpful in solving large-dimensional problems. For details, see Section 3.

Matrix manipulations are based on the transformations used in digital signal processing. They are suitable for solving large-dimensional problems in combination with graphic structure of data, such as DD. Details are presented in Section 4.

Taylor expansion is the analog of the Taylor series in mathematical analysis. Its *merits* are that the coefficients of an arithmetical polynomial admit representation in a convenient form for analysis by analogs of Boolean differences (as arithmetical derivatives).

Manipulations based on the algebra of corteges—the topic for discussion and investigation in this paper—underlie the Malyugin approach, which significantly differs from the first three approaches. Its *merits* are (*i*) algorithmically flexible manipulation over large-dimensional arithmetical expressions, (*ii*) new possibilities for realization of direct and inverse transformations, (*iii*) simplicity of design and analysis of LARs, and (*vi*) possibility of representation of any Boolean function by a finite LAR set. The last property is not inherent in any of the well-known method. The Malyugin method is described in Section 5.

These transformation methods differ in computational complexity and algorithmic realizability. The choice of a method depends on the concrete problem.

[3] In scientific literature, arithmetical transformation is often interpreted in terms of signal theory: a Boolean function is a representation of a signal in time domain and its corresponding arithmetical polynomial is a form of representation of signals in the spectral domain [15, 22].

[4] This classification does not include the well-known methods of *approximate* computation of coefficients of arithmetical polynomials via probabilistic transformations [14, 15].

3. TRANSFORMATION TO ALGEBRAIC FORM

Arithmetical transformation can be realized directly by representing every element in a logical circuit by an appropriate polynomial [23]. This is one of merits of the algebraic form.

3.1. Direct Transformation

Manipulations over arithmetical expressions are based on the linearity of arithmetical transformations. Let $\mathcal{A}\{f\}$ denote the arithmetical transform of a Boolean function f represented by $f = f_1 \vee f_2 \ldots \vee f_r$. Then the arithmetical analog of the Boolean function $\mathcal{A}\{f\}$ is constructed via transformation of subfunctions f_i

$$\mathcal{A}\{f\} = \mathcal{A}\{f_1 \vee f_2 \ldots \vee f_r\} = \mathcal{A}\{f_1\} + \mathcal{A}\{f_2\} \ldots + \mathcal{A}\{f_r\}.$$

Example 2. Using the linearity property, the trivial equalities $\overline{x} = 1 - x$, $x_1 \vee x_2 = x_1 + x_2 - x_1x_2$, $x_1x_2 = x_1x_2$, and $x_1 \oplus x_2 = x_1 + x_2 - 2x_1x_2$ can be generalized to functions. The Boolean functions $f_1 = x_1\overline{x}_2 \vee x_1\overline{x}_3$, $f_2 = x_1 \vee (x_2 \oplus x_3)$ and $f_3 = x_1 \oplus x_2$ are reduced to arithmetical form as

$$\mathcal{A}\{f_1\} = \mathcal{A}\{x_1\overline{x}_2\} + \mathcal{A}\{x_1\overline{x}_3\} - \mathcal{A}\{x_1\overline{x}_2\}\mathcal{A}\{x_1\overline{x}_3\} = x_1\overline{x}_2 + x_1\overline{x}_3 - x_1\overline{x}_2\overline{x}_3,$$
$$\mathcal{A}\{f_2\} = x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - 2x_2x_3 + 2x_1x_2x_3,$$
$$\mathcal{A}\{f_3\} = x_1 - 2x_1x_2 + x_2.$$

Since arithmetical transforms are linear, the outputs of the Boolean function f can be grouped as words (1), i.e., functions of every output can be represented by the weighted sum of transforms

$$\mathcal{A}\{f\} = \mathcal{A}\{2^{r-1}f_r\} + \ldots + \mathcal{A}\{2^1f_2\} + \mathcal{A}\{2^0f_1\} = 2^{r-1}P_r + \ldots + 2^1P_2 + 2^0P_1 = D. \qquad (3)$$

Example 3. Applying property (3) to the Boolean function in Example 2, we obtain

$$D = 2^0f_1 + 2^1f_2 + 2^2f_3 = 7x_1 + 6x_2 + 2x_3 - 10x_1x_2 - 4x_1x_3 - 4x_2x_3 + 3x_1x_2x_3.$$

In this example, three Boolean functions, each of which is represented by an arithmetical equivalent, are described by one (nonlinear) polynomial, i.e, in word-level form. This polynomial is obviously inferior in complexity to the initial Boolean functions. All methods of grouping of functions (numeration of outputs of the function as in Example 1) do not yield the linear expression (2). This is a motivation for developing new linearization methods.

3.2. Inverse Transformation

Since inverse transformation to algebraic form mostly requires an intuitive approach and hardly yields to algorithmization, it is not of much interest even for small-dimensional problems.

4. TRANSFORMATION TO MATRIX FORM

Here we examine two matrix arithmetical transformations [11]. Let $\mathbf{X}$ and $\mathbf{D}$ be the truth vector of a Boolean function f and the vector of coefficients of its arithmetical representation, respectively. The relationship between $\mathbf{X}$ and $\mathbf{P}$ is defined by two transformations

$$\mathbf{D} = \mathbf{A}_{2^n}\mathbf{X} \text{ (direct transformation)}, \qquad (4)$$
$$\mathbf{X} = \mathbf{A}_{2^n}^{-1}\mathbf{D} \text{ (inverse transformation)}, \qquad (5)$$

where $\mathbf{A}_{2^n}$ and $\mathbf{A}_{2^n}^{-1}$ are the matrices of direct and inverse arithmetical transformations, respectively.

4.1. Direct Transformation

Direct transformation (4) is applied to represent a Boolean function in arithmetical form defined by the coefficient vector $\mathbf{D}$.[5] The *arithmetical transformation matrix* A_{2^n} is formed by the rule

$$\mathbf{A}_{2^n} = \bigotimes_{i=1}^{n} \mathbf{A}_1, \quad \mathbf{A}_1 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix},$$

where A_1 is the *base matrix of direct transformation* and $\otimes$ denotes the Kronecker multiplication.

Example 4. The three-output Boolean function of Example 3 is transformed to an arithmetical polynomial as follows. First we form its truth vector with integral elements

$$\mathbf{X} = \underbrace{[\mathbf{X}_3|\mathbf{X}_2|\mathbf{X}_1]}_{\text{Truth matrix}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \\ 7 \\ 5 \\ 1 \\ 2 \end{bmatrix}.$$

Then applying the direct transformation (4), we obtain the coefficient vector $\mathbf{D}$ of the unknown arithmetical polynomial D

$$\mathbf{D} = \mathbf{A}_{2^3}\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \\ 7 \\ 5 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \\ 7 \\ 5 \\ 1 \\ 2 \end{bmatrix} \begin{matrix} \\ x_3 \\ x_2 \\ x_2x_3 \\ x_1 \\ x_1x_3 \\ x_1x_2 \\ x_1x_2x_3 \end{matrix}.$$

This example shows that matrix form is preferable to algebraic form in computations.

4.2. Inverse Transformation

Inverse transformation applied to the coefficient vector D (4) yields the truth vector $\mathbf{X}$. The *matrix of inverse arithmetical transformation* $A^{-1}(n)$ is formed according to the rule

$$\mathbf{A}_{2^n}^{-1} = \bigotimes_{i=1}^{n} \mathbf{A}_1^{-1}, \quad \mathbf{A}_1^{-1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix},$$

where A_1^{-1} is the *base matrix of inverse transformation*.

Example 5. Find the Boolean function corresponding to $LAR = 2 + 3x_1 + 5x_2$. The inverse transformation (5) generates the truth vector $\mathbf{X}$

$$\mathbf{X} = \mathbf{A}_{2^n}^{-1}\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 5 \\ 10 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

Consequently, $LAR = 2 + 3x_1 + 5x_2$ represents a Boolean function of four outputs $f_4 = x_1x_2$, $f_3 = x_1 \oplus x_2$, $f_2 = \overline{x}_1 \oplus x_1x_2$, and $f_1 = x_1 \oplus x_2$.

[5] The coefficient vector is also called the arithmetical spectrum of the Boolean function.

5. TRANSFORMATIONS IN CORTEGE ALGEBRA

A common drawback of these three methods of arithmetical transformations (Fig. 1) is the restricted possibilities for manipulations over functions in word-level representation for linearization. Widening these possibilities, Malyugin solved the following problem: given a many-output Boolean function, find (a) an algebra for effective manipulation over word-level structured data and (b) a method for linearization of arithmetical expressions in this algebra.

5.1. Initial Definitions

Without specifying the nature of an object (element), a cortege Cor_m is defined to be an ordered sequence of m elements $U_i, i = 1, 2, \ldots, m$:

$$U_m \odot U_{m-1} \odot \cdots \odot U_1 = Cor_m = \bigodot_{i=1}^{m} U_i, \tag{6}$$

where $\odot$ denotes the separator between elements. Often we write ${}^{A}Cor_m$ to denote that the cortege A contains m elements. In arithmetical logic, elements of a cortege may be constants 0 and 1, variables x_i, functions f_i (Boolean expressions), and (nonlinear) polynomials (in the general case), denoted in the sequel by AR_i.

Example 6. (i) $Cor = 0 \odot x_3 \odot 1$ is a cortege of constants, (ii) $Cor = f_3 \odot f_2 \odot f_1$ is a cortege of functions, (iii) $Cor = AR_3 \odot AR_2 \odot AR_1$ is a cortege of polynomials, and (iv) $Cor = (x_2 \vee x_3) \odot (x_1 \oplus x_2) \odot 0$ is a cortege of Boolean expressions and a constant.

The number of an element in a cortege is its address. The advantage of description by a cortege is the possibility for grouping and manipulation over data in word-level format. For example, $LAR = x_1 + x_2$ represents a Boolean function with two outputs x_1x_2 and $x_1 \oplus x_2$, which is defined by the cortege $x_1 + x_2 \Longleftrightarrow \underbrace{(x_1x_2)}_{\text{Higher-order bit}} \odot \underbrace{(x_1 \oplus x_2).}_{\text{Lower-order bit}}$ The *masking operator* is designed to extract the desired function from a cortege or LAR.

Definition 2. The masking operator

$$f_r = \Xi_r^m\{f\} \tag{7}$$

shows the position r of the function f_r in the cortege Cor_m of m functions $f_i,\ i = 1, 2, \ldots, m$.

Example 7. Functions in a cortege $x_1x_2 \odot (x_1 \oplus x_2)$ are "extracted" with a masking operator $x_1x_2 = \Xi_2^2\{x_1 + x_2\}$ and $x_1 \oplus x_2 = \Xi_2^1\{x_1 + x_2\}$.

5.2. Cortege Algebra

The main requirements imposed on the algebra used in the description and transformation of switching functions are *functional completeness* (distinguishable algebraic representation for a function), *flexibility* (reasonable complexity of data manipulation algorithms), and *realizability* (simplicity and obvious physical interpretation of the relationship with other data structures). Using these requirements, Malyugin extended the classical Boolean algebra to operations over corteges.

5.3. Sum of Corteges

We can find the sum of a cortege A of elements a_i and a cortege B of elements b_i in analogy with the design of a binary adder, whose primitive components are called half-adders.[6] Let us

[6] A binary half-adder computes the sum of two binary numbers: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$ (Fig. 2). The results of the first three operations are represented by only one bit. But if both operands are 1, the sum is represented by two bits, which correspond to *CARRY* and *SUM*.

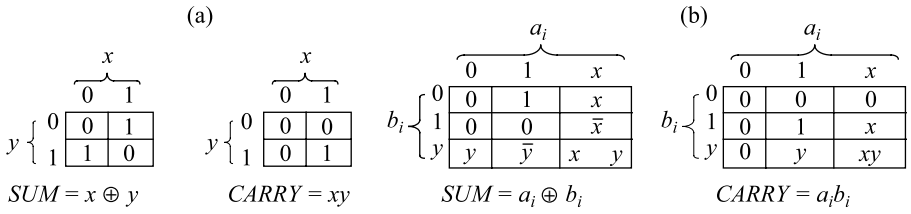

Fig. 2. Truth tables of a binary half-adder (a) and cortege half-adder (b) (8).

b_i \ a_i Carry	00	01	0x	10	11	1x	x0	x1	xx
0	0	1	x	1	0	$\overline{x}$	x	$\overline{x}$	0
1	1	0	$\overline{x}$	0	1	x	$\overline{x}$	x	1
y	y	$\overline{y}$	$x \oplus y$	$\overline{y}$	y	$\overline{x} \oplus y$	$x \oplus y$	$\overline{x} \oplus y$	y

b_i \ a_i Carry	00	01	0x	10	11	1x	x0	x1	xx
0	0	0	0	0	1	x	0	x	x
1	0	1	xy	y	1	$x \vee y$	xy	$x \vee y$	x
y	0	y	xy	y	y	$x \vee y$	xy	$x \vee y$	x

Fig. 3. Truth tables of a complete two-input cortege adder (9) (top table for *Sum* and bottom table for *Carry*).

design a *half-adder for corteges.* Since the elements of a cortege may be constants 0, 1, variables, or Boolean expressions, let us express the initial data as $a_i, b_i \in \{0, 1, x, y\}$, where x and y are Boolean variables. Addition of constants 0 and 1 differs from the functions of a binary half-adder only in the form of expressions $0+0 = 0 \odot 0$, $0+1 = 0 \odot 1$, $1+0 = 0 \odot 1$, and $1+1 = 1 \odot 0$. For the addition of constants and variables, there are four situations (a) $0 + x = 0 \odot x$, (b) $1 + x = x \odot \overline{x}$, (c) $y+0 = 0 \odot y$, and (d) $y+1 = 1 \odot \overline{y}$, since carry operation in the addition of $(1+x)$ and $(1+y)$ is formed only for $x = y = 1$. Finally, $x + y = (x \oplus y) \odot xy$, because $SUM = x \oplus y$ and $CARRY = xy$ in the binary half-adder. From the truth tables for a half-adder of corteges and a binary half-adder shown in Fig. 2 we arrive at

Assertion 1. *Binary half-adders and cortege half-adders have identical formal descriptions*

$$SUM = a_i \oplus b_i, \quad CARRY = a_i b_i. \tag{8}$$

Using half-adder (8), we can design a *complete adder of corteges.*[7] Addition of constants is implemented as in a binary adder, including the case $1 + 1 + 1 = 1 \odot 1$. The four cases considered above for the addition of constants 0 and 1 and variables x and y are similar to those for a cortege half-adder. The result of addition for other combinations of constants and variables is (i) $1 + x + 1 = 1 \odot x$, (ii) $1 + y + 1 = 1 \odot y$, and (iii) $x + 1 + y = 0 \odot (\overline{x} \oplus y)$ since $x + 1 + y = (x+1) + y = (x \odot \overline{x}) + y = (x \oplus \overline{x}y) \odot (\overline{x} \oplus y) = (x \vee y) \odot (\overline{x} \oplus y)$. Finally, the sum of three elements of a cortege for $x + x + y$ and $x + y + y$ is $x + x + y = (x + x) + y = (x \odot 0) + y = x \odot y$. From the truth tables of adders for the addition of binary numbers and corteges (Fig. 3), we arrive at

Assertion 2. *Formal descriptions of binary adders and cortege adders are identical*

$$SUM = a_i \oplus b_i \oplus CARRY_{i-1}, \quad CARRY_i = a_i b_i \vee CARRY_{i-1}(a_i \oplus b_i). \tag{9}$$

[7] A two-bit adder has two inputs for terms x and y and an input for carrying from the preceding digit $CARRY_{i-1}$. The formal model is defined by two relations $SUM = x \oplus y \oplus CARRY_{i-1}$ and $CARRY = xy + CARRY_{i-1}(x \oplus y)$.

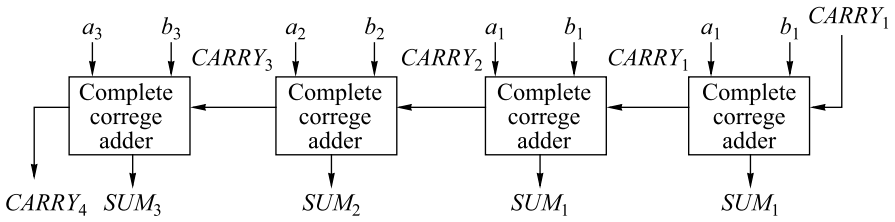

Fig. 4. A sequential adder of four-element corteges.

Interpretation of addition of corteges by an adder (9) is useful in designing algorithms for manipulations over corteges. For example, the analog of a parallel adder of corteges is the parallel binary adder, and the algorithm for sequential addition of corteges is derived from the architecture of the sequential binary adder (Fig. 4).

5.4. Product of Corteges

The product of two corteges is conveniently interpreted with a binary multiplier based on the accumulation of partial products. The product of two corteges ${}^{A}Cor_2 = x \odot 1$ and ${}^{B}Cor_2 = 0 \odot y$ is

$$\begin{array}{rccccc} \times & & x & \odot & 1 \\ & & 0 & \odot & y \\ \hline & & xy & & 1y \\ & 0 & 0 & & \\ \hline & 0 & \odot\ xy & \odot & y \end{array} .$$

Here AND-terms are partial products. The result ${}^{A}Cor_2 \times {}^{B}Cor_2 = 0 \odot xy \odot y$ is formed by accumulating partial products, carrying digits from right to left.

5.5. Properties of Cortege Algebra

Let three corteges ${}^{A}Cor_m$, ${}^{B}Cor_s$, and ${}^{C}Cor_k$ be given. To extend the Malyugin arithmetical logic, let us define a few axioms of cortege algebra (0 and 1 elements, complementation, commutativity, associativity, and distributivity). Using them, we can implement any cortege transformation.

Example 8. A cortege Cor_k is given. A cortege with inverse order of occurrence of elements is formed as follows. Since $Cor_k + \overline{Cor}_k = 2^k - 1$, we have $\overline{Cor}_k = 2^k - 1 - Cor_k$. The inverse of the cortege $Cor_2 = x_1 \odot x_1x_2$ is the cortege $\overline{Cor}_2 = 2^2 - 1 - Cor_2 = 3 - x_1 \odot x_1x_2 = (1 \odot 1) - (x_1 \odot x_1x_2) = 1 - x_1 \odot 1 - x_1x_2 = \overline{x}_1 \odot \overline{x_1}x_2$, where $3 = 1 \odot 1$, or in vector form

$$Cor_2 = x_1 \odot x_1x_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad \overline{Cor}_2 = \overline{x}_1 \odot \overline{x_1}x_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

5.6. Particular Cases

Particular cases of cortege operations are helpful in simplifying transformations and designing algorithms.

Addition of a constant and a function. Let us express the relation $x_1 + x_2 = x_1x_2 \odot (x_1 \oplus x_2)$ between a function f and a constant $a \in \{0, 1\}$ as

$$a + f = af \odot (a \oplus f). \tag{10}$$

Relation (10) cannot be used for manipulations over corteges since it holds only for single-bit operations. For $a = 0$, we have $0 + f = 0f \odot (0 \oplus f) = f$. Similarly, for $a = 1$ we obtain $1 + f = 1f \odot (1 \oplus f) = f \odot \overline{f}$. Let us find $a + f$ for $a > 1$. Let $a = 2$. Since $a = 2$ and the function f is represented as a cortege of $1 \odot 0$ and $f = 0 \odot f$, we have

$$\begin{array}{rrcccc} + & 2 & = & 1 & \odot & 0 \\ & f & = & 0 & \odot & f \\ \hline & 2+f & = & 1 & \odot & f \end{array} .$$

Multiplication of a constant by a function. In analogy with addition, for $a = 3 = 1 \odot 1$ we obtain $3 + f = f \odot \overline{f} \odot \overline{f}$. Since $0f = f$, $1f = f$, and $2 = 1 \odot 0$, the product of 2 and f is

$$\begin{array}{rrcccc} \wedge & 2 & = & 1 & \odot & 0 \\ & f & = & & & f \\ \hline & 2f & = & f & \odot & 0 \end{array} .$$

Technique of manipulations over corteges. **Example 9.** The sum and product of a function $f = x_1 \oplus x_2$ and a constant 5 are equal to $5 + f = 1 \odot (x_1 \oplus x_2) \odot (\overline{x_1 \oplus x_2})$ and $5f = (x_1 \oplus x_2) \odot 0 \odot (x_1 \oplus x_2)$, respectively, or in vector form

$$5 + f = 5 + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad 5f = 5 \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Since $x_1 \oplus x_2 = x_1 + x_2 - 2x_1x_2$, the result in arithmetical polynomial form is $5 + f = 5 + x_1 + x_2 - 2x_1x_2$, $5f = 5(x_1 + x_2 - 2x_1x_2)$.

The next example illustrates the technique of manipulations involved in the addition of a cortege Cor and a constant a, i.e., $Cor + a$.

Example 10. Given a cortege $Cor = (x_1 \vee x_2) \odot x_2$ and a constant $a = 1$, find a new cortege $Cor = \{(x_1 \vee x_2) \odot x_2\} + 1$. Let us apply the computation scheme

$$\begin{array}{rrcccc} + & Cor & = & x_1 \vee x_2 & \odot & x_2 \\ & 1 & = & 0 & \odot & 1 \\ \hline & Cor+1 & = & x_2 \odot x_1\overline{x}_2 & \odot & \overline{x}_2 \end{array} .$$

For the lower-order digit, we have $x_2 + 1 = x_2 \odot \overline{x}_2$. The next digit is $(x_1 \vee x_2) + 0 + x_2 = (x_1 \vee x_2) + x_2$. Using the relation $f_1 + f_2 = f_1 f_2 \odot (f_1 \oplus f_2)$, we obtain $(x_1 \vee x_2)x_2 \odot (x_1 \vee x_2) \oplus x_2 = x_2 \odot x_1\overline{x}_2$. The same result is obtained by computation in vector form

$$1 + Cor = 1 + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = x_2 \odot x_1\overline{x}_2 \odot \overline{x}_2.$$

The result of Example 10 is interpreted in terms of arithmetical logic as follows: the initial polynomial $AR = 2(x_1 \vee x_2) + x_2 = 2x_1 + 3x_2 - 2x_1x_2$ is transformed to the polynomial $AR + 1 = 2^2x_2 + 2^1x_1\overline{x}_2 + 2^0\overline{x}_2 = 4x_2 + 2x_1\overline{x}_2 + \overline{x}_2$.

5.7. Direct and Inverse Transformations

Direct arithmetical transformation is helpful in representing any Boolean function as an arithmetical polynomial with the help of cortege algebra. The initial function can be defined by Boolean expressions, a cortege, or a truth table.

Given an arithmetical polynomial AR, find its corresponding cortege Cor of Boolean functions, i.e., $\mathcal{A}^{-1}(AR) = Cor$. The solution algorithm consists of (a) representation of every term of the polynomial AR by a cortege (in the general case, the polynomial may contain 2^n terms, i.e., 2^n corteges), (b) specification of the general format m for all corteges, and (c) computation of the sum of corteges. Formally, the solution is guaranteed by

Theorem 1 ([9]). *A cortege of Boolean functions is uniquely estimated from a given arithmetical polynomial AR by the rule*

$$\mathcal{A}^{-1}(AR) = Cor_m = \sum_{i=0}^{2^n-1} \bigodot_{j=1}^{m} \tau_{ij}, \tag{11}$$

where τ_{ij} is the jth element in the ith cortege (term of the polynomial).

The proof of Theorem 1 is given in [9].

Example 11 (continuation of Example 5). To find the cortege of Boolean functions from the polynomial $LAR = 2 + 3x_1 + 5x_2$, let us express its terms, using (11), as corteges $2 = 1 \odot 0$, $3x_1 = x_1 \odot x_1$, and $5x_2 = x_2 \odot 0 \odot x_2$. Hence $LAR = (\overbrace{1 \odot 0}^{2}) + (\overbrace{x_1 \odot x_1}^{3x_1}) + (\overbrace{x_2 \odot 0 \odot x_2}^{5x_2})$ and

$$\begin{array}{cccccccccc}
+ & 2 & = & & & 1 & \odot & 0 & & \\
 & 3x_1 & = & & & x_1 & \odot & x_1 & & \\
\hline
+ & & & x_1 & \odot & \overline{x}_1 & \odot & x_1 & & \\
 & 5x_2 & = & x_2 & \odot & 0 & \odot & x_2 & & \\
\hline
 & & x_1x_2 & \odot & x_1 \oplus x_2 & \odot & \overline{x}_1 \oplus x_1x_2 & \odot & x_1 \oplus x_2 &
\end{array} .$$

Finally, $LAR = 2 + 3x_1 + 5x_2 = (\overbrace{x_1x_2}^{f_4}) \odot (\overbrace{x_1 \oplus x_2}^{f_3}) \odot (\overbrace{\overline{x}_1 \oplus x_1x_2}^{f_2}) \odot (\overbrace{x_1 \oplus x_2}^{f_1})$.

6. LINEARIZATION

As a result of linearization of word-level models, the number of nonzero elements is reduced to the number of variables (2). The LAR is mapped into a linear graph, i.e., a decision diagram. In certain well-known publications, the linear structure of data is determined using different properties of Boolean functions (logical circuits), which must be preliminarily identified. In 1952, Komamiya [17] used a LAR of the type $A_1 + A_2 + \ldots + A_n = d_m 2^m + d_{m-1} 2^{m-1} + \ldots + d_1 2 + d_0$ to describe adders, where A_i, $d_i \in \{0, 1\}$. For example, for $n = 2$ this relation describes a half-adder, and for $n = 3$ a total adder. In this section, we describe the so-called *naive* or direct approach to linearization, which dominated in investigations for a long time, and the Malyugin approach.

6.1. Linearization Condition

A Boolean function f of n variables with m outputs $f_0, f_1, \ldots, f_{m-1}$ is defined by a truth vector $[f^0 f^1 \ldots f^{2^n-1}]$ with integral elements f^j.

Theorem 2 (linearization condition [10, 27]). *An m-output Boolean function f of n variables admits representation by LARs if and only if the truth vectors f^j, $j \in \{0, 1, \ldots, 2^n - 1\}$, and f^{2^i}, $i \in \{0, 1, \ldots, n-1\}$, $j \neq \{0, 2^0, 2^1, \ldots, 2^{n-1}\}$, satisfy the equality*

$$f^j = \sum_{i=0}^{n-1} j_{n-i} f^{2^i} + \left(1 - \sum_{i=1}^{n} j_i f^0\right). \tag{12}$$

The proof of Theorem 2 is given in [10, 27].

This theorem formulates *necessary and sufficient conditions* for linearization of arithmetical word-level models of Boolean functions.

Example 12 (continuation of Example 1). The outputs of a half-adder are described by two functions $f_1 = [f_1^0 f_1^1 f_1^2 f_1^3] = [0110]$ (sum) and $f_2 = [f_2^0 f_2^1 f_2^2 f_2^3] = [0001]$ (carry). Let us take a fixed order for the outputs $[f_2\ f_1]$. The linearization condition (12) is satisfied since the equality $2(f_2^0 - f_2^1 - f_2^2 + f_2^3) = -f_1^0 + f_1^1 + f_1^2 - f_1^3$, i.e., $2(0 - 0 - 0 + 1) = -0 + 1 + 1 - 0$ holds for $n = 2$. By virtue of the truth vector $[f_2\ f_1] = [0\ 1\ 1\ 2]^T$, we can express the coefficient vector, using, for example, the direct matrix transformation (4): $\mathbf{D} = [0\ 1\ 1\ 0]$ is the corresponding polynomial $LAR = x_1 + x_2$. If the order of numeration for the outputs of the function is changed to $[f_1\ f_2]$, condition (12) is not satisfied and the half-adder is described by the linear polynomial $2x_1 + 2x_2 - 3x_1x_2$.

Let us state the constraints in the direct (naive) approach to AR linearization.

Theorem 3. *A single-output Boolean function can be uniquely represented by a LAR, which describes a new extended Boolean function with two outputs, one of which is the initial function.*

The proof of Theorem 3 is given in the Appendix.

According to Theorem 3, the *extended* Boolean function contains a garbage function. For example, applying Theorem 3 to the function $f = x_1 \vee x_2 = x_1 + x_2 - x_1x_2$, we obtain $LAR = x_1 + x_2 + 1$ for the cortege $(x_1 \vee x_2) \odot (1 \oplus x_1 \oplus x_2)$, where $1 \oplus x_1 \oplus x_2$ is a garbage function. The theorem on the LAR representation of a Boolean function of three variables is proved along similar lines. But serious difficulties are encountered in applying this method to functions of three or more variables.

6.2. LAR of Primitive Functions

Malyugin designed an elegant approach to linearization of arithmetical polynomials using cortege algebra [9].

Theorem 4 (LAR of primitive functions [9]). *AND, OR, and EXOR functions with n inputs admit representation by LAR*

$$n\text{-}AND \quad \bigwedge_{i=1}^{n} x_i^{\sigma_i} = \Xi_m^m \left\{ 2^{j-1} - n + \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

$$n\text{-}OR \quad \bigvee_{i=1}^{n} x_i^{\sigma_i} = \Xi_m^m \left\{ 2^{j-1} - 1 + \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

$$n\text{-}EXOR \quad \bigoplus_{i=1}^{n} x_i^{\sigma_i} = \Xi_m^1 \left\{ \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

where

$$j = \lceil \log_2 n \rceil + 1, \quad x_i^{\sigma_i} = \begin{cases} x_i, & \sigma_i = 0 \\ \overline{x}_i, & \sigma_i = 1, \end{cases} \tag{13}$$

and $\lceil a \rceil$ is the integral part of the number a.

The proof of Theorem 4 is given in [9].

Table 1. Corteges and LAR of primitive elements of the typical CAD IC library

Element	Polynomial	Cortege	LAR	Function
OR $x \vee y$	$x+y-xy$	$\begin{bmatrix}0\\1\\1\\1\end{bmatrix} \odot \begin{bmatrix}1\\0\\0\\1\end{bmatrix} = (x \vee y) \odot (\overline{x \oplus y})$	$1+x+y$	$\Xi_2^2\{1+x+y\}$
NOR $\overline{x \vee y}$	$1-x-y+xy$	$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \odot \begin{bmatrix}0\\1\\1\\0\end{bmatrix} = \overline{(x \vee y)} \odot (x \oplus y)$	$2-x-y$	$\Xi_2^2\{2-x-y\}$
AND xy	xy	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \odot \begin{bmatrix}0\\1\\1\\0\end{bmatrix} = xy \odot (x \oplus y)$	$x+y$	$\Xi_2^2\{x+y\}$
$NAND$ $\overline{xy}$	$1-xy$	$\begin{bmatrix}1\\1\\1\\0\end{bmatrix} \odot \begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \overline{xy} \odot (\overline{x \oplus y})$	$3-x-y$	$\Xi_2^2\{3-x-y\}$
$EXOR$ $x \oplus y$	$x+y-2xy$	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \odot \begin{bmatrix}0\\1\\1\\0\end{bmatrix} = xy \odot (x \oplus y)$	$x+y$	$\Xi_2^1\{x+y\}$

Remark on Theorem 4. (i) AND (OR) and EXOR functions correspond to the higher-order bit (m) and lower-order bit of number 1 in word-level format, which is determined by the masking operators Ξ_m^m and Ξ_m^1, respectively.

(ii) The term $1 \oplus x_i^{\sigma_i}$ takes either the value $1 \oplus x_i = \overline{x}_i$ ($\sigma_i = 0$) or the value $x_i \oplus \overline{x}_i = 1$ ($\sigma_i = 1$).

Table 1 illustrates the application of Theorem 4. In all cases, linearization requires one garbage function to be added. Moreover, the negation operation NOT corresponds to LAR $\Xi_1^1\{1-x\}$. Using Table 1, we can find LAR of other elements, for example,

$$\overline{x} \vee y = \Xi_2^2\{1+(1-x)+y\} = \Xi_2^2\{2-x+y\},$$
$$\overline{x} \oplus y = \Xi_2^1\{(1-x)+y\} = \Xi_2^1\{1-x+y\}.$$

LAR for 3-input elements are designed with Theorem 4 along similar lines:

$$LAR_{OR} = \Xi_3^3\{3+x+y+z\}, \quad LAR_{NAND} = \Xi_3^3\{6-x-y-z\},$$
$$LAR_{NOR} = \Xi_3^3\{4-x-y-z\}, \quad LAR_{EXOR} = \Xi_3^1\{x+y+z\},$$
$$LAR_{AND} = \Xi_3^3\{1+x+y+z\}, \quad LAR_{EXNOR} = \Xi_3^1\{1+x+y+z\}.$$

6.3. Linear Decision Diagrams

In this section, we apply the word-level DD technique to manipulate over arithmetical expressions. The linear word-level DD for representing a many-out Boolean function is called the *linear decision diagram* (LDD), i.e., a word-level decision diagram satisfying linearity conditions.

Definition 3. A linear decision diagram (LDD) is a graph (tree) for representing a linear arithmetical polynomial, whose terminal nodes are put into correspondence with the coefficients of

the linear arithmetical polynomial, and nodes correspond to variables and realize the arithmetical analog of the Davio expansion pD_A[8] (arithmetic positive Davio expansion) :

$$f = f_{x_i=0} + x_i(-f_{x_i=0} + f_{x_i=1}) \quad \text{is node,} \tag{14}$$

where $f_{x_i=0} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ and $f_{x_i=1} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ are the left and right branches, respectively.

Theorem 5. *Every combination circuit*[9] *with elements from the standard CAD IC library admits representation by a set of linear decision diagrams.*

The proof is implied by the possibility for structuring any logical circuit in the form of a finite number of levels, each of which is described by LARs and direct mapping of a LAR into an LDD (nodes define the operation pD_A and hanging vertices represent the LAR coefficients). The order of elements in a level must be fixed so that the finiteness property is preserved. In the general case, LAR and LDD are semi-canonical representations of logical functions.

The complexity of arithmetical models in the form of LAR and LDD is estimated with

Theorem 6. *An n-output element of the standard library is represented by an LDD with n nodes and* $(n + 1)$ *hanging vertices.*

The proof follows from the recurrent pD_A expansion of a LAR of n variables. Consequently, the complexity of representation of a multilevel circuit with G elements is $O(G)$.

Example 13. The linear polynomial $LAR = -3x_1 - 12x_2 + 17x_3 + 20$ is represented by a three-vertex LDD. Its variables can occur in any order. $LAR_{x_1=0}$ and $LAR_{x_1=1}$ are

$$LAR_{x_1=0} = -3 \times 0 - 12x_2 + 17x_3 + 20 \text{ left branch,}$$
$$LAR_{x_1=1} = -3 \times 1 - 12x_2 + 17x_3 + 20.$$

According to (14), the term $-f_{x=0} + f_{x=1}$ is equal to the constant $-LAR_{x_1=0} + LAR_{x_1=1} = -3$. Therefore, the hanging vertex is assigned the value -3 (Fig. 5). Since $LAR_{x_1=0}$ is not a constant and $LAR_{x_1=0} = -12x_2 + 17x_3 + 20$, we use the expansion (left and right branches, respectively)

$$LAR_{\substack{x_1=0\\x_2=0}} = 17x_3 + 20 \quad \text{and} \quad LAR_{\substack{x_1=0\\x_2=1}} = -12 + 17x_3 + 20.$$

Hence the next hanging vertex for $x_1 = 0$ and $x_2 = 0$ takes the value -12. Decomposition at the next level gives two hanging vertices with values 17 and 20, respectively.

7. COMPUTATION TECHNIQUE

7.1. Weighted Sum of LAR

The underlying principle of computation is the representation of a finite set of LAR_i by the weighted sum $2^{j_1}LAR_0 + 2^{j_2}LAR_1 + \ldots + 2^{j_r}LAR_r$, $j_1 < j_2 < \ldots < j_r$. The number of bits to represent a LAR is $j = \lceil \log_2(n) \rceil + 1$, where n is the number of bits in the representation of the maximal value of the LAR.

[8] A distinction is made between the Davio positive $f = f_0 \oplus x_1 f_2$ and Davio negative $f = \overline{x}_1 f_2 \oplus f_1$ expansions (Davio's results on AND-EXOR logical expressions are very useful in designing digital devices [28]) and Shannon expansion $f = \overline{x}_1 f_0 \oplus x_1 f_1$, where $f_0 = f(0, x_2, \ldots, x_n)$, $f_1 = f(1, x_2, \ldots, x_n)$, $f_2 = f_0 \oplus f_1$.

[9] In the general case, every memory circuit can be represented by LAR and LDD.

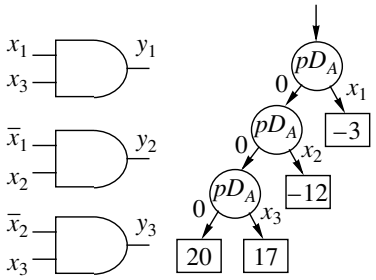

Fig. 5. Levels of a three-AND element circuit corresponding to $LAR = -3x_1 - 12x_2 + 17x_3 + 20$ and LDD.

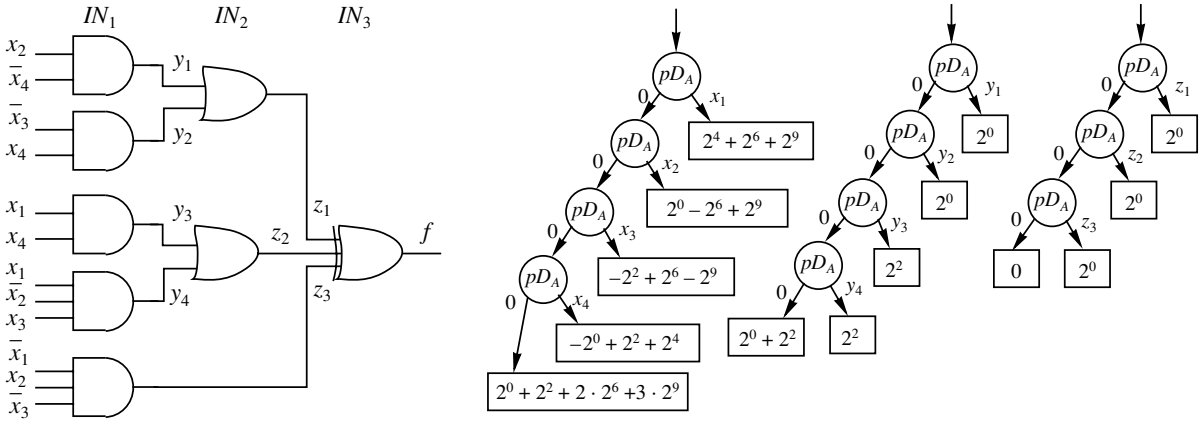

Fig. 6. An AND/OR/EXOR circuit (left) and its LDD system (right).

Example 14. Let us consider the levels of a three-element circuit (Fig. 5). The first element represents a LAR of weight $2^0 = 1$ and $j_1 = \lceil \log_2(2) \rceil + 1 = 2$ bits are required for its representation. The second and third LARs have weights 2^2 and 2^4 and require $j_2 = 2$ and $j_1 + j_2 = 4$ bits, respectively. Consequently, the circuit level in Fig. 5 is described by the LAR

$$LAR = 2^0(x_1 + x_2) + 2^2(\overline{x}_1 + x_2) + 2^4(\overline{x}_2 + x_3)$$
$$= 2^0(x_1 + x_2) + 2^2(1 - x_1 + x_2) + 2^4(1 - x_2 + x_3) = -3x_1 - 12x_2 + 17x_3 + 20.$$

The resultant LAR is the linear model for the lth level IN_l of the circuit with n inputs and r outputs. Representation of the ith LAR requires a memory $J_i = j_1 + \ldots + j_{i-1} + j_i = J_{i-1} + j_i$, where $i \in \{1, 2, \ldots, r\}$ and $J_0 = 0$. Computation with this model is reduced to computing the weighted sum LAR_i, $i \in \{1, \ldots, r\}$:

$$LAR = IN_l = \sum_{i=1}^{r} (2^{J_i} LAR_i), \tag{15}$$

where $j_i = \lceil \log_2 n \rceil + 1$ is the number of bits required to represent LAR_i.

7.2. Representation of Multilevel Circuits by a System of LDD

Using Theorem 5 and Example 13, let us describe a method for designing linear models of LAR and LDD for any logical circuit consisting of element of the standard CAD IC library.

Example 15. An AND/OR/EXOR circuit (Fig. 6) is given. Using the LAR model of elements in Table 1, let us construct an LDD model. The LAR of the first circuit level is

$$\begin{array}{lll} LAR_1 = & 2^0(1+x_2-x_4) & \Rightarrow \text{element } 1 \\ + & 2^2(1+x_4-x_3) & \Rightarrow \text{element } 2 \\ + & 2^4(x_1+x_4) & \Rightarrow \text{element } 3 \\ + & 2^6(2+x_1-x_2+x_3) & \Rightarrow \text{element } 4 \\ + & 2^9(3-x_1+x_2-x_3) & \Rightarrow \text{element } 5\,. \end{array}$$

Weights required for describing the second circuit level and the corresponding LARs are $y_1 = \Xi^2\{LAR_1\}$, $y_2 = \Xi^4\{LAR_1\}$, $y_3 = \Xi^6\{LAR_1\}$, and $y_4 = \Xi^9\{LAR_1\}$, $LAR_2 = 2^0(1+y_1+y_2) + 2^2(1+y_3+y_4)$. Similarly, LAR of the third circuit level are $z_1 = \Xi^2\{LAR_2\}$, $z_2 = \Xi^4\{LAR_2\}$, $z_3 = \Xi^{12}\{LAR_1\}$ and $LAR_2 = z_1 + z_2 + z_3$. Finally, the circuit output is $f = \Xi^1\{LAR_3\}$. The LDD model is shown in Fig. 6.

Even in this simple 8-element circuit, coefficients take values as high as 2^{12}. Industrial circuits contain millions of elements. This is the motivation for developing new approaches to designing linear models.

8. LARGE WEIGHT COEFFICIENTS

Arithmetical models *cannot be used in CAD IC due to the large values of coefficients* even in circuits with 10–20 inputs. The well-known approaches based on modular arithmetic [30] only partly solve this problem, which we now we solve using the properties of the coefficients of LARs.

8.1. Properties of Coefficients

Hypothesis. Since every logical circuit is designed from a bounded number of standard elements with known LAR (Table 1), we may successfully apply the properties of linear models based on simple linear models of primitive elements.

Theorem 7. *The lth level IN_l of a logical circuit with n inputs and r elements admits representation by a LAR*

$$IN_l = W_0 + W_{x_1}x_1 + \ldots + W_{x_n}x_n, \tag{16}$$

where the weights of hanging vertices W_0 and W_{x_i}, $i = 1, \ldots, n$, are

$$W_0 = a_{0,1}2^{J_1} + \ldots + a_{0,r}2^{J_r}, \text{ and } W_{x_i} = a_{i,1}2^{J_1} + \ldots + a_{i,r}2^{J_r}, \tag{17}$$

$a_{i,t} \in \{0, +1, -1\}$ and $a_{0,t}$ are positive integers, $J_1 = j_0 = 0$, $J_t = j_0 + \ldots + j_{t-1}$, $j_t = \lceil \log_2 n_t \rceil + 1$, and n_t is the number of inputs of the function obtained by extending y_t for deriving $LAR(y_t)$.

The proof of Theorem 7 is given in the Appendix.

Corollary 1. *The parameters J_i, except for $J_0 = 0$, are defined by the difference* $\mathrm{M} = J_i - J_{i-1}$ *since the coefficients of hanging vertices are the sum of weights 2^{J_i}, where J_i is the sum of j_l $(l < i)$.*

Theorem 7 and Corollary 1 yield a coding algorithm based on the substitution of the values of hanging vertices by the respective codes. Let us describe this algorithm with an example (for details, see [12, 31]). Consider the LDD in Fig. 5. The first hanging vertex is assigned the value -3. Since the weight of the LAR of the first element is 2^0 ($J_1 = 0$) and x_1 is contained without negation

Index	9	8	7	6	5	4	3	2	1
Weight	2^{24}	2^{21}	2^{18}	2^{15}	2^{12}	2^{9}	2^{6}	2^{3}	2^{0}
Code	01	01	00	00	11	11	11	01	00

Fig. 7. Coding of the hanging vertex in LDD (Example 16).

($b_1 = 1$). The weight of LAR for the second element is 2^2 ($J_2 = 2$) and x_1 is contained with negation ($b_1 = 1$). Consequently,

$$W_{x_1} = b_1 2^{J_1} + b_2 2^{J_2} = 12^0 - 12^2 = -3. \tag{18}$$

Relation (18) shows that the number -3 is uniquely represented by four attributes b_1, J_1, b_2, and J_2. Obviously, b_i and J_i require far less memory than the weights of hanging vertices for storage. Indeed, since b_i can take only three values $b_i \in \{0, 1, -1\}$, two bits are sufficient for its code, for example, $0 \Rightarrow 00$, $1 \Rightarrow 01$, and $-1 \Rightarrow 11$.

According to Theorem 7 and Corollary 1, the coefficients of hanging vertices are not arbitrary integers. Consequently, continuing the example, we find that it suffices to store only $J_1 = 0$ and $J_2 - J_1 = 2$. Therefore, for the first hanging vertex, we must store the parameters $b_1 = 01$, $J_1 = 000$, $b_2 = 11$, and $J_2 - J_1 = j_2 = 010$.

Now we consider the second hanging vertex. Its weight is $W_{x_2} = b_1 2^{J_1} + b_2 2^{J_2} = 1 \times 2^2 - 1 \times 2^4 = -12$, which can be represented by four parameters $b_1 = 01$, $J_1 = 010$, $b_2 = 11$, and $j_2 = 010$. Similarly, the weight of the third hanging vertex is $W_{x_3} = b_1 2^{J_1} + b_2 2^{J_2} = 1 \times 2^0 + 1 \times 2^4 = 20$. But it is sufficient to store $b_1 = 01$, $J_1 = 000$, $b_2 = 01$, and $J_2 - J_1 = 100$. The parameters of the free hanging vertex are $a_1 = 01$, $J_1 = 000$, $b_2 = 01$, and $J_2 - J_1 = 010$, which correspond to the weight $W_0 = a_1 2^{J_1} + a_2 2^{J_2} = 1 \times 2^0 + 1 \times 2^2$.

Example 16. The code of the value $0 + 2^3 - 2^6 - 2^9 - 2^{12} + 0 + 0 + 2^{21} + 2^{24}$ of the hanging vertex is shown in Fig. 7. The coefficient of the linear model attains the value 2^{24} and is coded by 18 bits.

9. EXPERIMENTAL RESULTS

Results of experiments conducted with large-dimensional test circuits are reported in [12]. Every circuit (including, circuits with multivalued gates) is shown to be representable by a finite set of LAR and LDD. A fragment of experiments is listed in Table 2, where TEST, #IN, #OUT, and #G denote the name, number of inputs, outputs, and gates of the circuit[10], respectively. The columns EDIF,

Table 2. Memories for LDD, EDIF, ISCAS models

TEST	#IN	#OUT	#G	EDIF	LDD	ISCAS	LDD
C1355	41	32	546	273 018	16 133	59 573	16 850
C1908	33	25	880	230 507	13 098	78 574	22 491
C2670	233	108	1193	400 707	32 395	110 025	36 305
C3540	50	24	1669	503 457	34 744	145 359	54 857
C5315	178	163	2307	903 899	88 965	220 596	107 392
C6288	32	32	2416	1 387 230	71 476	255 406	68 572
C7552	207	109	3512	1 236 066	146 057	311 939	271 970

[10] Circuits of the LGSynth 93 base, http://zodiac.cbl.ncsu.edu/CBL_Docs/lgs93.html, were used as test specimens in experiments.

ISCAS, and LDD show memory (in bytes) for the traditional representation of a circuit in EDIF, ISCAS, and LDD formats. Clearly, LDD models require 9–25 times less memory than traditional models. For example, the combination circuit C7552 with 207 inputs, 109 outputs, and 3512 gates requires 1 236 066 and 162 222 bytes for storage in EDIF and ISCAS formats, respectively, i.e., 22 and 10 times less memory for storage than in LDD format. Verification of linear circuit models is still an unresolved problem.

10. ARITHMETICAL LOGIC IN NANOTECHNOLOGY

The *deterministic* models described in this paper may not find direct utilization in future technologies due to the stochastic behavior of nanostructures [32]. This stimulates the study of *probabilistic* and *information* models for computation. Arithmetical logic is the boundary case of the so-called *probabilistic logic*. Merekin [33] was probably the first to prove that the *reliability function of the switching circuit is the generalization to arithmetical description of the circuit*. Malyugin used this property to study the reliability of switching circuits with arithmetical polynomials [34].

Example 17. Let signals x_1 and x_2 appear at the input of an OR gate independently with probabilities $p_1 = p(x_1)$ and $p_2 = p(x_2)$, respectively. The output y of the OR gate is either x_1 or x_2. Let $p_1 = 0.8$ and $p_2 = 0.9$. Then the correct output must be expected with probability $p = 1 - (1 - p_1)(1 - p_2) = p_1 + p_2 - p_1p_2 = 0.8 + 0.9 - 0.8 \times 0.9 = 0.98$.

Table 3. Deterministic and probabilistic models of typical gates

Gate	Conjunctor	Disjunctor	Mod 2 adder
Deterministic model			
Function	$y = x_1x_2$	$y = x_1 \vee x_2$	$y = x_1 \oplus x_2$
Polynomial	$y = x_1x_2$	$y = x_1 + x_2 - x_1x_2$	$y = x_1 + x_2 - 2x_1x_2$
LAR	$\Xi_2^2\{x_1 + x_2\}$	$\Xi_2^2\{1 + x_1 + x_2\}$	$\Xi_2^1\{x_1 + x_2\}$
Probabilistic model			
Model	$p = p_1p_2$	$p = p_1 + p_2 - p_1p_2$	$p = p_1 + p_2 - 2p_1p_2$

Taking $p_1 = p_2 = 1$, let us transform the probabilistic model into a deterministic model. Obviously, the output of the OR element is $y = x_1 + x_2 - x_1x_2$, which is the arithmetical description of the OR element. Table 3 shows the probabilistic characteristics of AND, OR, and EXOR gates.

11. CONCLUSIONS

Malyugin's theorems jointly with the latest results in arithmetical logic and modern CAD IC tools are effective in linearizing arithmetical forms of representation of not only Boolean functions, but also multivalued logical functions, as demonstrated in [12]. Possibly, these results may be useful in future technologies and other problems, viz., in artificial intelligence systems.

ACKNOWLEDGMENTS

I express my indebtedness to S. Yanushkevich, G. Duek (Canada), P. Dziurzanski, T. Lub (Poland), Sasao (Japan), M. Karkovsky (USA), B. Falkowski (Singapore), O. Fin'ko (Russia), and R. Drechsler (Germany) for their valuable discussion and assistance at different stages of preparation of the paper. I also express my thanks to the referees for their remarks.

APPENDIX

Proof of Theorem 3. Each of 16^2 possible pairs of 16 Boolean functions of two variables uniquely corresponds to an arithmetical expression also containing LAR (2). Indeed, two functions are defined by four values $f = (f_0, f_1, f_2, f_3)$ and $f' = (f'_0, f'_1, f'_2, f'_3)$ and represent 256 possible two-output Boolean functions f and f' or an integer function with values $(2f_0 + f'_0, 2f_1 + f'_1, 2f_2 + f'_2, 2f_3 + f'_3)$. These functions correspond to the arithmetical expression $2^1 f + 2^0 f'$ and satisfy condition (2) if

$$2f_3 + f'_3 = -(2f_0 + f'_0) + (2f_1 + f'_1) + (2f_2 + f'_2). \tag{19}$$

Expression (19) holds only for 44 two-outputs functions out of 256 functions. But each of 16 subfunctions occurs at least once at the first and second places. Consequently, any one-output function of two variables can be represented by LAR via extension, i.e., addition of a function in word level format such that the result is a LAR.

Proof of Theorem 7. Let the output of the first element belonging to the level be described by the expression $LAR(y_1) = 2^1 f_2 + 2^0 f_1$. Since $AR(f_1) = a_{0.1} + a_{1.1}x_2 + a_{2.1}x_1 + a_{3.1}x_1x_2$ and $AR(f_2) = a_{0.2} + a_{1.2}x_2 + a_{2.2}x_1 + a_{3.2}x_1x_2$, we obtain

$$\begin{aligned} LAR(y_1) &= 2^1(a_{0.2} + a_{1.2}x_2 + a_{2.2}x_1 + a_{3.2}x_1x_2) + 2^0(a_{0.1} + a_{1.1}x_2 + a_{2.1}x_1 + a_{3.1}x_1x_2) \\ &= (2^1a_{0.2} + 2^0a_{0.1}) + (2^1a_{1.2} + 2^0a_{1.1})x_2 + (2^1a_{2.2} + 2^0a_{2.1})x_1 + (2^1a_{3.2} + 2^0a_{3.1})x_1x_2. \end{aligned}$$

Discarding nonlinear terms (all components are linear), we obtain $LAR(y_1) = (2^1a_{0.2} + 2^0a_{0.1}) + (2^1a_{1.2} + 2^0a_{1.1})x_2 + (2^1a_{2.2} + 2^0a_{2.1})x_1$. Let $W_0 = 2^1a_{0.2} + 2^0a_{0.1}$, $W_{x_1} = 2^1a_{1.2} + 2^0a_{1.1}$, and $W_{x_2} = 2^1a_{2.2} + 2^0a_{2.1}$. Then $LAR(y_1) = W_0 + W_{x_1}x_1 + W_{x_2}x_2$ (weights are determined with lower-order digits of the values of the function in LAR). Applying the same approach, we obtain LARs of other elements.

REFERENCES

1. Bryant, R. and Chen, Y., Verification of Arithmetic Functions Using Binary Moment Diagrams, *Proc. Design Automat. Conf.*, 1995, pp. 535–541.
2. Drechsler, R., *Formal Verification of Circuits*, Boston: Kluwer, 2000.
3. Lai, Y., Pedram, M., and Vrudhula, S., EVBDD-based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition, *IEEE Trans. CAD Integrated Circuits Syst.*, 1994, vol. 13, no. 8, pp. 959–975.
4. Minato, S., *Binary Decision Diagrams and Applications for VLSI CAD*, Boston: Kluwer, 1996.
5. Aiken, H.H., *The Annals of the Computation Laboratory of Harvard University, Synthesis Electron. Comput. Control Circuits*, Cambridge: Harvard University, 1951, vol. XXVII.
6. Huang, C.-Y. and Cheng, K.-T., Using Word-Level ATPG and Modular Arithmetic Constraint—Solving Techniques for Assertion Property Checking, *IEEE Trans. CAD Integrated Circuits Syst.*, 2001, vol. 20, no. 3, pp. 381–391.
7. Rahardia, S. and Falkowski, B.J., Fast Linearly Independent Arithmetic Expansion, *IEEE Trans. Comput.*, 1999, vol. 48, no. 9, pp. 991–999.
8. Malyugin, V.D., Realization of Boolean Functions by Arithmetical Polynomials, *Avtom. Telemekh.*, 1982, no. 4, pp. 84–93.
9. Malyugin, V.D., Realization of a Cortege of Boolean Functions by Linear Arithmetical Polynomials, *Avtom. Telemekh.*, 1984, no. 2, pp. 114–121.
10. Antonenko, V.M., Ivanov, A.A., and Shmerko, V.P., Linear Arithmetical Forms of k-valued Logical Functions and Their Realization by Systolic Arrays, *Avtom. Telemekh.*, 1995, no. 3, pp. 139–155.
11. Shmerko, V.P., Synthesis of Arithmetical Forms of Boolean Functions through Fourier Transformation, *Avtom. Telemekh.*, 1989, no. 5, pp. 134–142.

12. Dziurzanski, P., Malyugin, V., Shmerko, V., and Yanushkevich, S., Linear Models of Circuits of Multivalued Elements, *Avtom. Telemekh.*, 2002, no. 6, pp. 99–119.
13. Falkowski, B.J., A Note on the Polynomial Form of Boolean Functions and Related Topics, *IEEE Trans. Comput.*, 1999, vol. 48, no. 8, pp. 860–863.
14. Jain, J., Bitner, J., Fussell, D.S., and Abraham, J.A., Probabilistic Verification of Boolean Functions, in: *Formal Methods in Syst. Design*, Boston: Kluwer Academic, 1992, vol. 1, pp. 61–115.
15. Thornton, M.A., Dreschler, R., and Miller, D.M., *Spectral Techniques in VLSI CAD*, London: Kluwer Academic, 2002.
16. Nechiporuk, E.I., Design of Circuits via Linear Transformations of Variables, *Dokl. Akad. Nauk SSSR*, 1958, vol. 123, no. 4, pp. 610–612.
17. Stankovic, R.S. and Sasao, T., Discussion on the History of Research in Arithmetic and Reed–Muller Expressions, *IEEE Trans. CAD of Integrated Circuits and Syst.*, 2001, vol. 20, no. 9, pp. 1177–1179.
18. Eris, E. and Muzio, J.C., Spectral Testing of Circuit Realisations Based on Linearizations, *IEE Proc., Part. E*, 1986, vol. 133, no. 2, pp. 73–78.
19. Karpovsky, M.G. and Moskalev, E.S., Utilization of Autocorrelation Characteristics for the Realization of Systems of Logical Functions, *Avtom. Telemekh.*, 1970.
20. Kolpakov, A.V. and Latypov, R.Kh., Approximate Algorithm for Minimization of Binary Decision Diagrams based on Linear Transformations of Variables, *Avtom. Telemekh.*, 2004, no. 6.
21. Stankovic, R.S. and Astola, J.T., Some Remarks on Linear Transform of Variables in Adders, *Proc. 5th Int. Workshop on Applications of the Reed–Muller Expansions in Circuit Design*, Mississipi State Univ., 2001, pp. 294–302.
22. Heidtmann, K.D., Arithmetic Spectrum Applied to Fault Detection for Combinational Networks, *IEEE Trans. Comput.*, 1991, vol. 40, no. 3, pp. 320–324.
23. Papaioannou, S.G., Optimal Test Generation in Combinational Networks by Pseudo-Boolean Programming, *IEEE Trans. Comput.*, 1977, vol. 26, pp. 553–560.
24. Yanushkevich, S., Arithmetical Canonical Expansions of Boolean and MVL Functions as Generalized Reed–Muller Series, *Proc. IFIP WG 10.5 Workshop on Appl. Reed–Muller Expansions in Circuit Design*, Japan, 1995, pp. 300–307.
25. Yanushkevich, S., Systolic Algorithm for Designing Arithmetical Polynomial Forms of k-valued Functions of Logical Algebra, *Avtom. Telemekh.*, 1994, no. 12, pp. 128–142.
26. Akers, S.B., On a Theory of Boolean Functions, *SIAM*, 1959, vol. 7, no. 4, pp. 487–498.
27. Artyukhov, V.L., Kondrat'ev, V.N., and Shalyto, A.A., Realization of Boolean Functions by Arithmetical Polynomials, *Avtom. Telemekh.*, 1988, no. 4, pp. 138–147.
28. Davio, M.J., Deschamps, P., and Thayse, A., *Discrete and Switching Functions*, New York: McGraw-Hill, 1978.
29. Malyugin, V.D., *Parallel'nye logicheskie vychisleniya posredstvom arifmeticheskikh polinomov* (Parallel Logical Computation through Arithmetical Polynomials), Moscow: Nauka, 1997.
30. Fin'ko, O.A., Realization of Large-Dimensional Systems of Boolean Functions by Modular Arithmetical Methods, *Avtom. Telemekh.*, 2004, no. 6.
31. Yanushkevich, S., Dziurzanski, P., and Shmerko, V., Word-Level Models for Efficient Computation of Multiple-Valued Functions: LAR, *IEEE 32th Int. Symp. Multiple-Valued Logic*, Boston, 2002, pp. 202–208.
32. Yamada, T., Kinoshita, Y., Kasai, S., Hasegawa, H., and Amemiya, Y., Quantum-Dot Logic Circuits Based on Shared Binary Decision Diagram, *Jpn. J. Appl. Phys.*, 2001, vol. 40, part 1, no. 7, pp. 4485–4488.
33. Merekin, Yu.V., Arithmetical Forms of Expression for Boolean Functions and Their Application in Circuit Reliability Computations, *Vychisl. Syst.*, 1963, vol. 7.
34. Malyugin, V.D., Reliability of Switching Circuits, *Avtom. Telemekh.*, 1964, no. 9, pp. 1375–1383.

This paper was recommended for publication by P.P. Parkhomenko, a member of the Editorial Board