

Representation of Logical Circuits by Linear Decision Diagrams with Extension to Nanostructures¹

P. Dziurzanskii*, V. P. Shmerko**, and S. N. Yanushkevich**

**Strzelin Technical University, Strzelin, Poland*

***University of Calgary, Calgary, Canada*

Received December 16, 2003

Abstract—The *LinearDD* system, which first permitted in logical design the representation of a logical circuit (combination circuit or a memory circuit) as two- and three-dimensional linear decision diagrams, is described. It is compatible with standard data formats (EDIF, ISCAS85, etc.) and designing tools, such as hardware description languages (HDL). This system is extended to circuit designing by nanotechnologies.

1. INTRODUCTION

Binary decision diagrams (BDD) for representing Boolean functions were designed by Akers in 1978 [1]. Ten year later, in 1986, Bryant designed a reduced ordered BDD (ROBDD) [2] and described its capabilities for data manipulations in large-dimensional graphic media. The result of manipulations, an optimized ROBDD, can be realized with elementary multiplexers, i.e., ROBDD is mapped into hardware description languages according to the scheme

Function (circuit) \iff Model in ROBDD form \iff Realization.

The dimension of a decision diagram (number of nodes) is exponentially bounded by the number of variables. But reasonable (polynomial) time and dimension can be attained under certain conditions. The variables ordering method greatly affects the dimension of ROBDD [2, 3]. Determination of the optimal order for variables is an NP-difficult problem. The next generation of decision diagrams, i.e., word-level decision diagrams (WDD), has considerably widened the dimension of problems [4, 5] due to processing of data in word-level format. One way of extending the possibilities of WDD consists in linearization of diagrams. Partial linearization of WDD jointly with graphic decomposition has been studied in depth. A linear WDD containing as many nodes as variables is the *boundary* case of WDD and represents a narrow class of functions. Malyugin described a method of extending this class via the so-called *linear* arithmetical logic [6] so that any logical circuit, even a memory circuit, is represented by a system of linear WDDs (LDD) [7, 8]. The underlying idea is the construction of *linear* arithmetical polynomials (LAR) that are directly mapped into an LDD, which is a *semi-canonical* data structure. Its inherent property is the compact representation of logical circuits compared to other formats used for this purpose. There is also a generalized LDD for logical circuits on many-valued elements [9, 10].

The *LinearDD* system (a program packet) realizes the representation of word-level data under linear boundary conditions. What matters here is that an ROBDD is converted into an LDD system

Circuit \iff ROBDD(WDD) model \iff LDD model \iff Realization.

¹ This work was supported in part by the NATO Collaborative Linkage Grant PST/CLG 979071 and Natural Sciences and Engineering Research Council of Canada (NSERC), project no. 239025-02.

The *LinearDD* system is effective, at least, for solving the following problems.

Problem 1. Circuit prototyping. Fast circuit prototyping includes three main stages, viz., designing, FPGA-programming, and debugging. The advantages of LDD models in transforming netlists to a data format suitable for prototyping. At this design stage, a file of hardware description language and C-codes used in FPGA programming, and a file for target emulation are generated.

Problem 2. Compact circuit representation. The LDD format, as a rule, requires less memory than other formats since the LDD size grows linearly with the number of elements in the netlist.

Problem 3. Intermediate format in data structure manipulations. The LDD format can be used as an intermediate format for generating circuits from VHDL description (a high-level language) for simplifying description and reducing design time, since regular LDD structures can be used at lower design levels. In prototyping systems, in particular, distributed and modular systems, with FPGA programmable devices, this property of LDD is very attractive.

Problem 4. Verification of circuits. Linear decision diagrams find restricted use for verification as they are quasi-canonical, though this is not a hurdle in many well-known verification methods.

In this paper, we shall show that an LDD can be imbedded in three-dimensional cubes of special structure, \mathcal{N} -hypercubes. Such a topological structure is reflected in the graph of a logical circuit corresponding to the main topological characteristics of many prospective technologies, in particular, *single electron* and *quantum dot array* nanotechnologies [11, 12]. Using \mathcal{N} -hypercubes, we can represent any circuit in a three-dimensional space.

2. THE *LinearDD* SYSTEM: ITS DESIGN AND CAPABILITIES

The *LinearDD* system primarily operates with linear arithmetical models based on linear arithmetical logic. Arithmetic polynomial representation of logical functions admits the description of a many-output function in word-level format.

Definition 1. A linear arithmetical polynomial (linear arithmetical expression or LAR) of a Boolean function f of n variables x_1, \dots, x_n is an expression with $(n + 1)$ integral coefficients d_0, d_1, \dots, d_n

$$LAR = d_0 + \sum_{i=1}^n d_i x_i = d_0 + d_1 x_1 + \dots + d_n x_n. \tag{1}$$

Definition 2. An LDD is a decision diagram into which a LAR is mapped: the nodes of LAR correspond to a positive Davio expansion pD_A , and the hanging vertices to the coefficients of the LAR. The LAR-to-LDD mapping is expressed as

$$LDD(d_0, d_1, \dots, d_n) \iff LAR.$$

The *LinearDD* system is helpful in representing any m -level circuit with levels $IN_i, i = 1, 2, \dots, m$ with elements of the standard primitive library. This system uses two data structures (models), i.e., a set of LARs (algebraic form) and a set of LDDs (graphic form).

Model 1. Let a logical circuit be defined by m levels

$$\begin{aligned} IN_1 &: \text{ inputs } x_j, & \text{ outputs } y_t, \\ IN_2 &: \text{ inputs } y_t, & \text{ outputs } s_l, \\ & \dots\dots\dots \\ IN_m &: \text{ inputs } u_p, & \text{ outputs } z_n. \end{aligned}$$

These levels correspond to the set of LAR_i :

$$\begin{aligned} LAR_1 &= d_0 + d_1x_1 + \dots + d_{q_1}x_{q_1}, \\ LAR_2 &= d'_0 + d'_1y_1 + \dots + d'_{q_2}y_{q_2}, \\ &\dots\dots\dots \\ LAR_m &= d''_0 + d''_1u_1 + \dots + d''_{q_m}u_{q_m}. \end{aligned}$$

Since a LAR is uniquely mapped into an LDD, “level-by-level” description is used in

Model 2. A set of LDDs represents a logical circuit

$$\begin{aligned} LDD_1(d_0, d_1, \dots, d_{q_1}) &\iff LAR_1, \\ LDD_2(d'_0, d'_1, \dots, d'_{q_2}) &\iff LAR_2, \\ &\dots\dots\dots \\ LDD_m(d''_0, d''_1, \dots, d''_{q_m}) &\iff LAR_m, \end{aligned}$$

where LAR_i corresponds to the i th level IN_i , $i = 1, 2, \dots, m$, of the circuit.

The underlying principles of *LinearDD* modeling are *compatibility* and *extension*. This system is compatible with standard tools of logical designing with three functions: *circuit transformation*, *format transformation*, and *library functions over primitives* (Table 1). This system is based on a unified internal format for transforming an LDD to standard formats. What matters here is that the *LinearDD* system is extended by the modular architecture principle. For example, extension of functions for designing nanoelements and devices is achieved only by including a module.

Circuit transformation. A multilevel logical circuit (even a memory circuit) is represented by a system of LDDs (in LDD format).

Format transformation. The initial circuit is defined in EDIF, PLA, BLIF, and Verilog formats. The internal format is converted into an EDIF, BLIF, Verilog, VHDL, XNF file or ROBDD format.

Library functions. The system contains a library of LDDs for simple 2- and 3-input circuits, and libraries of standard macroelements. These primitive are realized in three-dimensional space (3D) as a 3D module of primitives (Table 2).

Table 1. Possibilities of the *LinearDD* system

	TRANSFORMATION
Circuit-to-LDD	A logical circuit is transformed to LDD format
LDD-to-Circuit	LDD format is transformed to a logical circuit
BDD-to-LDD	A BDD is transformed to LDD format
LDD-to-Netlist	An LDD is transformed to netlist
LDD-to-EDIF	An LDD is transformed to EDIF format
LDD-to-XNF	An LDD is transformed to XNF format
LDD-to-VHDL	An LDD transformed to VHDL format
	LIBRARIES
2D LDD-Gate	A library of memoryless primitives
2D LDD-FF	A library of memory primitives
2D LDD-Macro	A library of macroelements

Table 2. Extension of the *LinearDD* system

3D LDD-Gate	A library of 3D-primitives for nanotechnologies
LDD verification	Functional verification of circuit realization
NN LDD	Computation with neural networks based on LDD
KN LDD	Representation of knowledge by LDD

The *LinearDD* system is based on the principles that permit the use of LDDs for modeling 3D data structures, verification of circuits, and use of artificial intelligence. An LDD is mapped into a 2D VLSI topological space with Verilog, VHDL, and other tools, which are used for input data description and formation of netlists. For example, the XNF-netlist is the output of Xilinx system for the FPGA design, and the ASIC system generates specifications in ASIC-topology.

3. MASKING TECHNIQUE

Arithmetical expressions underlying the design of the *LinearDD* system are *canonical* representations of logical functions. They are alternatives of the SOP, POS, and Reed–Muller forms of representation for Boolean functions [13, 14]. They admit the description of a multi-output function in word-level format. For example, the coefficient vector $D = [12101000]^T$ in (1) corresponds to $LAR = 1 + x_1 + x_2 + 2x_3$. *Linearization* consists of transformation of a nonlinear arithmetical expression into a linear expression, i.e., LAR. For example, the functions $f_1 = x_1 \oplus x_2 = x_1 + x_2 - 2x_1x_2$ and $f_2 = x_1x_2$ can be uniquely represented by the linear expression $LAR = 2^1 f_2 + 2^0 f_1 = 2^1 x_1x_2 + 2^0(x_1 + x_2 - 2x_1x_2) = x_1 + x_2$. Changing the order of numeration of functions, we obtain the nonlinear expression $AR = 2x_1 + 2x_2 - 3x_1x_2$. Consequently, expression (1) is a canonical representation and ordering of functions in a word is a decisive factor for linearization.

This example shows that the required functions must be “extracted” from the respective positions in the word with a mask. For this purpose, we must know the number of the binary digit of the value of the function. In the *LinearDD* packet, linear data structures are manipulated with the so-called *masking operator*. More exactly, a LAR represents a system of Boolean functions, which are subdivided into *information* and *garbage* functions. Information functions are initial functions, for which a LAR is constructed. Garbage functions do not carry any useful information and are used in the word-level format for the sake of linearization. The positions of information and garbage functions in a word are essential for linearization. Therefore, the positions of all functions must be known in advance and garbage functions must be hidden. Unfortunately, such linear models can be derived only for a narrow class of functions. This class contains elementary Boolean functions of two and three variables.

Definition 3 ([7]). A masking operator identifies the $(\xi - 1)$ th bit, $\xi \in \{1, \dots, k\}$, in the binary representation of a number $A = 2^{k-1}a_{k-1} + 2^{k-2}a_{k-2} + \dots + 2^{\xi-1}a_{\xi-1} + \dots + 2^0a_0$

$$\Xi^\xi\{A\} = \left\lfloor \frac{A}{2^{\xi-1}} \right\rfloor \pmod{2} = a_{\xi-1}, \tag{2}$$

where $\lfloor a \rfloor$ is the largest integer less than or equal to a . For example, the second bit ($\xi = 3$) in the binary code of the number $A = 7$ is identified as $a_2 = \lfloor 7/2^2 \rfloor \pmod{2} = 1$.

Example 1. A circuit with three AND elements G_1 , G_2 , and G_3 is shown in Fig. 1. Using Table 3, let us write the LAR of these elements as $LAR_{G_1} = x_1 + x_2$, $LAR_{G_2} = x_3 + x_4$, and

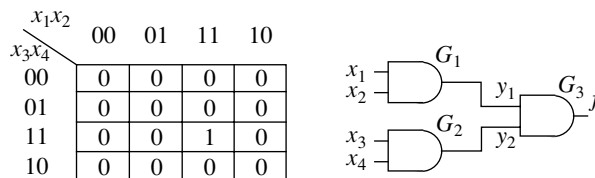


Fig. 1. A two-level circuit (Example 1).

Table 3. LAR models of logical primitives

Primitive	$\langle \text{PRIMITIVE} \rangle \iff \text{LAR}$	Mask
AND	$\bigwedge_{i=1}^n x_i^{\sigma_i} \iff 2^{j-1} - n + \sum_{i=1}^n (\sigma_i + (-1)^{\sigma_i} x_i)$	$\Xi^j \{ \text{LAR} \}$
OR	$\bigvee_{i=1}^n x_i^{\sigma_i} \iff \text{LAR} = 2^{j-1} - 1 + \sum_{i=1}^n (\sigma_i + (-1)^{\sigma_i} x_i)$	$\Xi^j \{ \text{LAR} \}$
EXOR	$\bigoplus_{i=1}^n x_i^{\sigma_i} \iff \text{LAR} = \sum_{i=1}^n (\sigma_i + (-1)^{\sigma_i} x_i)$	$\Xi^1 \{ \text{LAR} \}$

Primitive	Two inputs	Three inputs
OR	$\Xi^2 \{ 1 + x + y \}$	$\Xi^3 \{ 3 + x + y + z \}$
NOR	$\Xi^2 \{ 2 - x - y \}$	$\Xi^3 \{ 4 - x - y - z \}$
AND	$\Xi^2 \{ x + y \}$	$\Xi^3 \{ 1 + x + y + z \}$
NAND	$\Xi^2 \{ 3 - x - y \}$	$\Xi^3 \{ 6 - x - y - z \}$
EXOR	$\Xi^1 \{ x + y \}$	$\Xi^1 \{ x + y + z \}$
EXNOR	$\Xi^1 \{ 1 + x + y \}$	$\Xi^1 \{ 1 + x + y + z \}$

$LAR_{G_3} = y_1 + y_2$. The level-by-level description of the circuit, according to model (1), contains a first level $LAR_1 = x_1 + x_2 + 4x_3 + 4x_4$ and a second level $LAR_2 = y_1 + y_2$. The value of the output f is extracted with a masking operator

$$f = \Xi^2 \{ LAR_2 \} = \Xi^2 \{ y_1 + y_2 \} = \Xi^2 \{ \Xi^2 \{ LAR_1 \} + \Xi^4 \{ LAR_1 \} \}.$$

The operator produces the leading bit from two-bit values of LAR_{G_1} , LAR_{G_2} , and LAR_{G_3} .

4. LIBRARY OF LINEAR MODELS OF PRIMITIVES

The library of 2D-elements is based on linear arithmetical models shown in Table 3 [6, 9], where the parameters j and $x_i^{\sigma_i}$ are defined by the equalities

$$j = \lceil \log_2 n \rceil + 1, \quad x_i^{\sigma_i} = \begin{cases} x_i, & \text{if } \sigma_i = 0 \\ \bar{x}_i, & \text{if } \sigma_i = 1. \end{cases} \quad (3)$$

Here $\lceil a \rceil$ denotes the least integer less than or equal to a . LAR models of primitive elements for any number of inputs are formed from Table 3.

Example 2. A linear model for a circuit with three primitives of three variables ($n = 3$) $f_1 = x_1 x_2 \bar{x}_3$, $f_2 = x_1 \vee \bar{x}_2 \vee \bar{x}_3$, and $f_3 = x_1 \oplus x_2 \oplus \bar{x}_3$ is designed with Table 3 as

$$\begin{aligned} x_1 x_2 \bar{x}_3 &= \Xi^3 \{ 2^{3-1} - 3 + x_1 + x_2 + \bar{x}_3 \} = \Xi^3 \{ 2 + x_1 + x_2 - x_3 \}, \\ x_1 \vee \bar{x}_2 \vee \bar{x}_3 &= \Xi^3 \{ 2^{3-1} - 1 + x_1 + \bar{x}_2 + \bar{x}_3 \} = \Xi^3 \{ 5 + x_1 - x_2 - x_3 \}, \\ x_1 \oplus x_2 \oplus \bar{x}_3 &= \Xi^1 \{ x_1 + x_2 + \bar{x}_3 \} = \Xi^1 \{ 1 + x_1 + x_2 - x_3 \}, \end{aligned}$$

since, according to (3), $j = \lceil \log_2 3 \rceil + 1 = 2 + 1 = 3$ and $\bar{x} = 1 - x$.

5. CIRCUIT MODELS IN THE FORM OF LINEAR DECISION DIAGRAMS

The *LinearDD* system uses WDD-type decision diagrams for designing and manipulating arithmetical expressions. The WDD is an LDD if the linearity conditions for LAR are satisfied. The

level of a circuit is represented by a LAR and then mapped into an LDD. The number of LDDs is equal to the number of levels in the circuit. The Davio expansion pD_A

$$f = f_{\bar{x}} + x(f_x - f_{\bar{x}}) \tag{4}$$

is used at the nodes of LDD. Applying expansion (4) to a LAR, we obtain two subfunctions $f_{\bar{x}}$ and $(f_x - f_{\bar{x}})$, which do not depend on the expansion variable x : the subfunction $(f_x - f_{\bar{x}})$ is always a constant and equal to the coefficient of x in the LAR.

Definition 4. An LDD is the DD obtained by mapping a LAR; nodes correspond to the expansion pD_A and hanging vertices to the coefficients of the LAR.

The system designs LDD with the help of LAR representations of logical functions

$$\text{Primitive} \iff \text{LAR model} \iff \text{LDD model} \iff \text{Realization.}$$

For example, the AND function x_1x_2 is extended to the 2-output function $f' = x_1x_2$ and $f = x_1 \oplus x_2 = x_1 + x_2 + 2 - 2x_1x_2$. Therefore, $LAR = 2^1 f' + 2^0 f = 2x_1x_2 + x_1 + x_2 - 2x_1x_2 = x_1 + x_2$. The AND function is extracted with a masking operator.

Definition 5. Every m -level logical circuit with fixed order of elements is uniquely represented by a system of m LDDs as

$$\text{Circuit} \iff \text{LAR model} \iff \text{LDD model} \iff \text{Realization.}$$

The *validity* of the model is proved by the fact that each of m circuit levels containing r two-input elements of a fixed library is described by one LAR, since every element can be expressed as a LAR. The fixed order of elements guarantees the uniqueness of the LAR-to-LDD mapping.

5.1. Design of an LDD for a Given Circuit Level

Let us consider a m -level logical circuit, in which the l th level, $l \in (1, \dots, m)$, contains r two-input elements of the standard library. The output y_1 of the first element is described as

$$LAR(y_1) = 2^1 f_2 + 2^0 f_1$$

by extending f_2 , where $y_1 = f_1$. Other elements of the level are described along similar lines. For example, the output y_2 of the second element is $LAR(y_2) = 2^3 f_4 + 2^2 f_3$. Consequently, this level is described by a linear model of the type

$$LAR = LAR(y_1) + LAR(y_2) + \dots + LAR(y_r), \tag{5}$$

i.e., the LAR describes the circuit level as the sum of weighted LARs for every element y_1, \dots, y_r . Obviously, weights may take very large values. For example, for a level containing 100 elements, the maximal weight is 2^{200} . Our system uses a special code found from the solution of the problem of large weight coefficients [7, 8]. Therefore, we can design linear models for large circuits.

Model 3. The l th level of a circuit with n inputs x_1, \dots, x_n and r elements is described by

$$LAR = W_0 + W_{x_1}x_1 + \dots + W_{x_n}x_n, \tag{6}$$

where the values of hanging vertices W_0 and W_{x_i} , $i = 1, \dots, n$, are formed according to the rule

$$W_0 = a_{0,1}2^{J_1} + \dots + a_{0,r}2^{J_r}, \quad W_{x_i} = a_{i,1}2^{J_1} + \dots + a_{i,r}2^{J_r}, \tag{7}$$

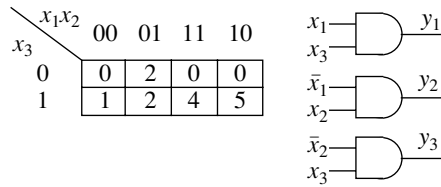


Fig. 2. Representation of a circuit level by an integral data structure (Example 3).

$a_{i,t} \in \{0, +1, -1\}$ and $a_{0,t}$ are positive integers, and

$$J_1 = j_0 = 0, \quad J_t = j_0 + \dots + j_{t-1}, \quad j_t = \lceil \log_2 n_t \rceil + 1, \tag{8}$$

where n_t is the number of inputs in the extended function.

Proof. Let the output of the first element in the circuit 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 $LAR(y_1) = 2^1(a_{0,2} + a_{1,2}x_2 + a_{2,2}x_1 + a_{3,2}x_1x_2) + 2^0(a_{0,1} + a_{1,1}x_2 + a_{2,1}x_1 + a_{3,1}x_1x_2) = (2^1 a_{0,2} + 2^0 a_{0,1}) + (2^1 a_{1,2} + 2^0 a_{1,1})x_2 + (2^1 a_{2,2} + 2^0 a_{2,1})x_1 + (2^1 a_{3,2} + 2^0 a_{3,1})x_1x_2$. All terms in this expression are LARs. Hence the result $LAR(y_1)$ is also a LAR. Since the nonlinear part is eliminated, we obtain $LAR(y_1) = (2^1 a_{0,2} + 2^0 a_{0,1}) + (2^1 a_{2,2} + 2^0 a_{2,1})x_2 + (2^1 a_{1,2} + 2^0 a_{1,1})x_1$. Let $W_0 = 2^1 a_{0,2} + 2^0 a_{0,1}$, $W_{x_1} = 2^1 a_{1,2} + 2^0 a_{1,1}$, and $W_{x_2} = 2^1 a_{2,2} + 2^0 a_{2,1}$. Then $LAR(y_1) = W_0 + W_{x_1}x_1 + W_{x_2}x_2$. Applying this expression to other elements of the level, we obtain LAR (6).

Corollary 1. The l th level IN_l of a circuit with n inputs x_1, \dots, x_n and r elements is represented by an LDD with n nodes corresponding to variables and $(n + 1)$ hanging vertices labelled by the values of the coefficients $W_0, W_{x_1}, \dots, W_{x_n}$ of the LAR_l .

Proof. The validity of the model is implied by Definition (5).

Example 3. Design a LAR model for the level of a logical circuit with three elements $y_1 = x_1x_3$, $y_2 = \bar{x}_1x_2$, and $y_3 = \bar{x}_2x_3$ (Fig. 2, right). First, using relation (6), let us derive a system of LARs

$$\begin{aligned} LAR(y_1) &= 2^1 f_2 + 2^0 f_1 = 2^0(x_1 + x_3), \\ LAR(y_2) &= 2^3 f_4 + 2^2 f_3 = 2^2(1 - x_1 + x_2), \\ LAR(y_3) &= 2^5 f_6 + 2^4 f_5 = 2^4(1 - x_2 + x_3), \end{aligned}$$

where $f_1 = x_1 \oplus x_3$, $f_2 = y_1$, $f_3 = \bar{x}_1 \oplus x_2$, $f_4 = y_2$, $f_5 = \bar{x}_2 \oplus x_3$, and $f_6 = y_3$ are garbage functions. Now, according to (5) and (6), $LAR = LAR(y_1) + LAR(y_2) + LAR(y_3) = W_0 + W_{x_1}x_1 + W_{x_2}x_2 + W_{x_3}x_3$. Since $j_1 = j_2 = \lceil \log_2 2 \rceil + 1 = 2$, $J_1 = j_0 = 0$, $J_2 = j_0 + j_1 = 0 + 2 = 2$, and $J_3 = j_0 + j_1 + j_2 = 4$,

$$\begin{aligned} W_0 &= 0 \times 2^{J_1} + 1 \times 2^{J_2} + 1 \times 2^{J_3} = 2^2 + 2^4, \\ W_{x_1} &= 1 \times 2^{J_1} - 1 \times 2^{J_2} + 0 \times 2^{J_3} = 2^0 - 2^2, \\ W_{x_2} &= 0 \times 2^{J_1} + 1 \times 2^{J_2} - 1 \times 2^{J_3} = 2^2 - 2^4, \\ W_{x_3} &= 1 \times 2^{J_1} + 0 \times 2^{J_2} + 1 \times 2^{J_3} = 2^0 + 2^4. \end{aligned}$$

Finally, $LAR = 2^2 + 2^4 + (2^0 - 2^2)x_1 + (2^2 - 2^4)x_2 + (2^0 + 2^4)x_3$ (Fig. 2, left).

Model 3 is used in the *LinearDesign* system as follows. The hanging vertices are subdivided into two types, viz., W_{x_i} and W_0 . The vertex values are determined from relation (7) with regard

for the property of weights W_{x_i} (8), $J_1 = j_0$, $J_2 = j_0 + j_1$, $J_r = j_0 + \dots + j_{r-1}$. In the system, the weight (of a terminal vertex) is represented by a slightly modified zero-suppressed BDD [15], denoted by *ZBDD.

Model 4. *The i th hanging vertex of weight W_{x_i} or W_0 (7) in an LDD can be represented by a *ZBDD.*

Proof. The weights of the i th hanging vertex W_{x_i} and hanging vertex W_0 contain the exponent of number $J_t = j_0 + j_1 + \dots + j_{t-1}$. Let the nodes of *ZBDD have the coefficients $2^{j_1}, 2^{j_2}, \dots, 2^{j_{t-1}}$. Then the path from the root to the hanging vertex of the diagram is described in terms of weights as $2^{J_1} = 2^{j_0} \Rightarrow 2^{J_2} = 2^{j_0} \times 2^{j_1} = 2^{j_0+j_1}, \Rightarrow 2^{J_r} = 2^{j_0} \times \dots \times 2^{j_{r-1}} = 2^{j_0+\dots+j_{r-1}}$, since the weights of nodes in this path are multiplied. The resultant weights are multiplied by $a_{i,t}$. Therefore, the weight of the t th hanging vertex is $a_{i,t}2^{J_t} = a_{i,t}2^{j_0+\dots+j_{t-1}}$. The final result is equal to the sum of weights thus obtained. Consequently, the structure of the linear *ZBDD corresponds to the weight of W_{x_i} or W_0 (7).

The values of hanging vertices in the LDD are represented by the data structure

```
int m_nLevel <the tree containing the node>;
int m_nIndex <the number of the given hanging vertex in LDD>;
int m_nCompValue <the value of the hanging vertex for a given input>;
vector<TerminalElement*> m_vecTerminalElements;
map<int,int> m_mTerminalElements.
```

The vector *m_vecTerminalElements* is of variable size and carries information on a terminal vertex. The structure *TerminalElement* contains components for storing weights and values of one hanging (terminal) vertex. Since the weight of a hanging vertex is formed from an associative table *m_mTerminalElements*, the time complexity of determination of this value is logarithmic.

6. DESIGN OF AN LDD FROM A NETLIST

For the *LinearDesign* system to be compatible with logical programming tools, netlist formats, in particular, Electronic Data Interchange Format (EDIF), Intl Symp. on Circuits and Systems (ISCAS), and Berkeley Logic Interchange (BLIF) formats are used. Compatibility with hardware description languages, like Verilog, is also an essential attribute of logical designing tools.

6.1. LDD Models of Combination Circuits

The underlying principle of the *LinearDesign* system is level-by-level linearization of a circuit. Such a description of a m -level circuit gives a system of m LDDs. Let the l th LDD (LDD_l) correspond to the l th level of a circuit, $l = 1, 2, \dots, m$. Then the model of a m -level combination circuit with r outputs f_1, \dots, f_r is a system of LARs

$$f_1 = \Xi^{\alpha_1} \{LAR_{s_1}\}, \dots, f_r = \Xi^{\alpha_r} \{LAR_{s_r}\}, \quad (9)$$

where $s_1, \dots, s_r \in \{1, \dots, m\}$ and $LAR_l = LAR(\Xi^{\beta_1} \{LAR_{k_1}\}, \dots, \Xi^{\beta_q} \{LAR_{k_q}\})$ for $l \in \{2, \dots, m\}$ and $k_1, \dots, k_q \in 1, \dots, l-1$.

Example 4. Representing the circuit in Fig. 3a in LAR format by relation (6), we obtain three LARs for the levels IN_1, IN_2 , and IN_3

$$\begin{aligned} LAR_1 &= W_0 + W_{x_1}x_1 + W_{x_2}x_2, \\ LAR_2 &= W_0 + W_yy + W_{x_3}x_3, \\ LAR_3 &= W_0 + W_zz + W_{x_4}x_4, \end{aligned}$$

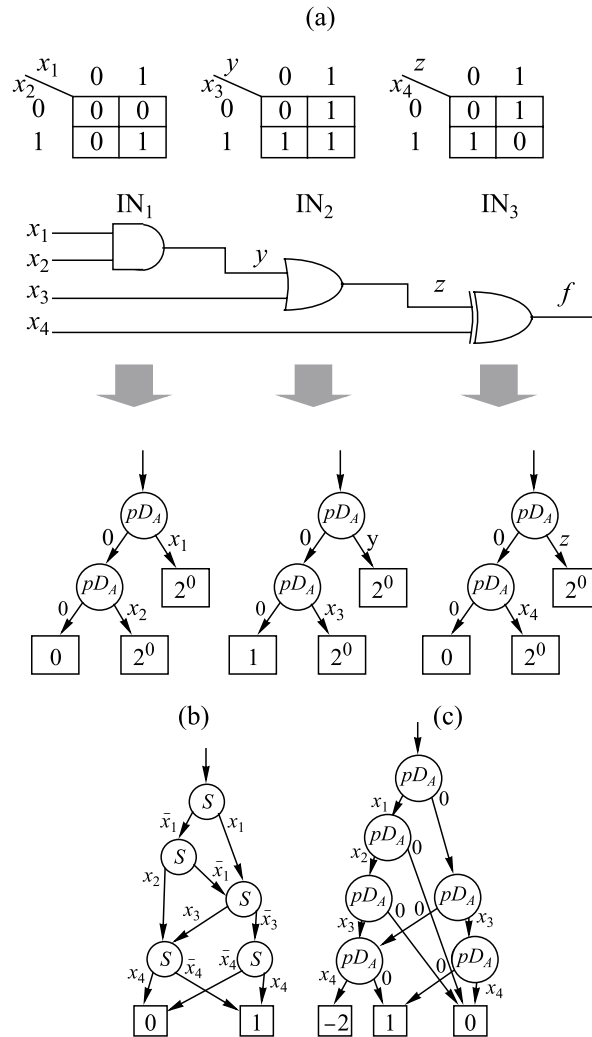


Fig. 3. An LDD model of a 3-level combination circuit (a) described by an ROBDD (b) and WDD (c) (Example 9).

where $y = \Xi^2\{LAR_1\}$ and $z = \Xi^2\{LAR_2\}$. Using the library of models (Table 3), we obtain $LAR_1 = x_1 + x_2$, $LAR_2 = 1 + y + x_3$, and $LAR_3 = z + x_4$. The values of functions are determined from the relations

$$\begin{aligned}
 f &= \Xi^1\{LAR_3\} = \Xi^1\{z + x_4\} = \Xi^1\{\Xi^2\{LAR_2\} + x_4\} \\
 &= \Xi^1\{\Xi^2\{\Xi^2\{LAR_1\} + 1 + x_3\} + x_4\} = \Xi^1\{\Xi^2\{\Xi^2\{x_1 + x_2\} + 1 + x_3\} + x_4\}.
 \end{aligned}$$

For the sake of comparison, Figs. 3b and c also show the ROBDD and WDD of this circuit.

An algorithm for designing LDD models is given in Table 4. It produces a set of unrelated LDDs. Data are transferred via an LDD by an algorithm based on the “postbox” principle or through a buffer for temporary storage of data similar to systolic arrays [16].

6.2. Memory Circuits

In designing LDD models for memory and feedback circuits, first all feedbacks must be identified and then ignore them by a special procedure *buildLDD(circuit)* (Table 5). This algorithm uses

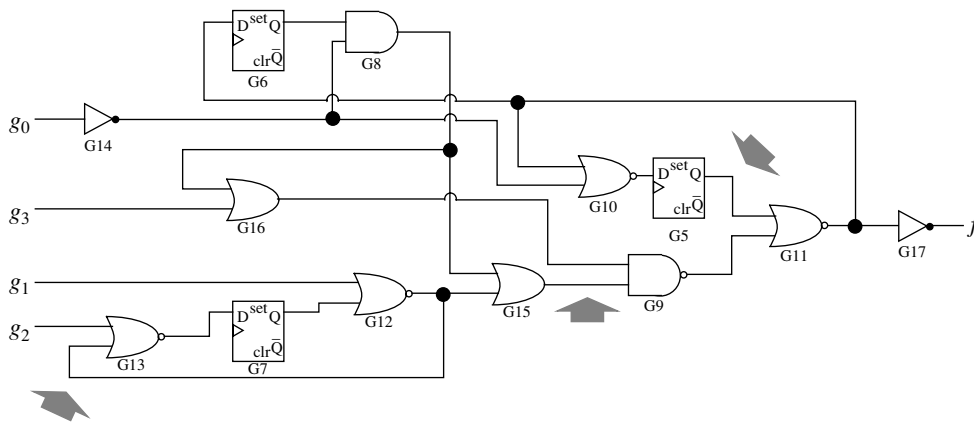


Fig. 4. Recognition of feedbacks (marked by arrows) in the s27a circuit by a DFS algorithm.

the depth first search (DFS) method for identifying feedbacks in input and output connections [17]. Upon completion of topological ordering of connections, the LDD design procedure is applied to every level of the combination circuit (Table 4). The following algorithm is used in the packet.

Assume that the DFS algorithm has recognized three feedbacks G5 with G11, G12 with G13, and G15 with G9 in the s27 circuit based on the ISCAS'89 family² (Fig. 4). Disregarding these connections, we find that the circuit consists of six levels and is representable by a system of six LDDs with 19 nodes. The triggers G5, G6, and G7 have additional feedbacks, which do not affect topological permutation.

Table 4. An algorithm for designing LDD models (left) and an algorithm for applying a “postbox” for data transfer between diagrams (right)

<pre> procedure buildLDD(<i>circuit</i>): {for (∀ level <i>l</i> in network) {construct LDD for level <i>l</i>; for (∀ gate <i>t</i> in level <i>l</i>) compute <i>weight_t</i> of element <i>t</i>; for (∀ gate <i>t</i> in level <i>l</i>) for (∀ input <i>in</i> of gate <i>t</i>) if (<i>in</i> is not included in LDD for level <i>l</i>) then add a new <i>pD_A</i> node and a variable <i>t_{in}</i> with input <i>in</i> of element <i>t</i>; assign weight <i>weight_t</i>; to the hanging vertex starting from the node <i>t_{in}</i>; }} </pre>	<pre> for (∀ terminal value <i>t</i> of the <i>l</i>-th LDD) {<i>tmp</i> ← 0; for (∀ node <i>i</i> in the level <i>l</i>) if (<i>i</i>-th node has coefficient <i>a_{i,t}</i> ≠ 0) then <i>tmp</i> ← <i>tmp</i> + <i>x_i</i> × <i>a_{i,t}</i>; <i>tmp</i> ← <i>tmp</i> + <i>a_{i,t}</i>; assign value to the current node <i>tmp</i> } </pre>
---	--

Table 5. An algorithm for designing an LDD model for data transfer between diagrams

```

procedure buildLDDSeq(circuit):
{ for (∀ circuit inputs in)
  apply DFS(in) and classify the branches;
  Apply topological sorting;
  Continue with algorithm buildLDD(circuit); }
    
```

² ISCAS'89 Benchmark Information. CAD Benchmarking Lab's Home Page. <http://www.cbl.ncsu.edu>.

6.3. Computation Complexity

The theoretical upper bound for the decision diagram dimension is exponentially proportional to the number of variables. In practice, however, a polynomial dependence is observed for certain classes of functions [2]. Moreover, the diagram dimension perceptibly depends on the order chosen for variables. Determination of the optimal order for variables is an NP-difficult problem.

Lemma 1. *An n -input element is represented by an LDD with n nodes and $(n + 1)$ hanging vertices.*

The proof follows from the definition of an LDD, which describes a LAR with $(n + 1)$ coefficients.

Corollary 2. *The level of a circuit with N inputs and p n -input elements is represented by one LDD with N nodes and $(N + 1)$ hanging vertices. If certain elements of the circuit have a common input, then $N \leq p \times n$.*

Theorem 1. *The time complexity of the algorithm of design of a set of linear decision diagrams for a multi-level combination circuit is $O(G)$, where G is the number of elements in the circuit.*

Proof. Let an l -level circuit consist of G elements. The algorithm described in Table 5 has three cycles. The first cycle involves l iterations, where l is the number of levels in the circuit. The second cycle consists of G_l iterations, where G_l is the number of elements in the l th level. The third cycle depends on the number of inputs of an individual element. Since this number is a constant, the number of iterations is $\sum_l G_l = G$, and the complexity is bounded by $O(G)$.

A sequential circuit is linearized in time $O(G + E)$, where G is the number of elements and E is the number of connections between them. This follows from the fact that the complexity of the DFS procedure is $O(G + E)$ [17]. Since the DFS procedure removes all feedbacks in the circuit, use is made of an algorithm for designing diagrams for combination circuits, whose complexity, according to Theorem 1, is $O(G)$.

Theorem 2. *The complexity of the algorithm for transforming a logical circuit into a set of linear decision diagrams is $O(G)$, where G is the number of elements in the circuit.*

Proof. Let an l -level circuit consist of G elements. Every level containing G_l n -input elements is described by one LDD with $n G_l$ nodes and $n (G_l + 1)$ end vertices. Therefore, the circuit is represented by a set of LDDs with a total number of nodes, equal to $n \times G$, and total number of hanging vertices, equal to $n \times G + l$. Since l is bounded by $O(G)$ and the maximal value of n is a constant (usually, 2 or 3), the memory size for storing an LDD is expressed in complexity by $O(G)$.

6.4. Experimental Estimates for the Dimension of LDDs

A series of experiments were conducted to compare the time required to construct an LDD and the size of the LDD generated by the *LinearDesign* system with the best results reported in papers and descriptions of program packets, particularly, CUDD³ and TUD⁴. The algorithms were tested with standard bases of logical circuits LGSynth'93 (in EDIF and ISCAS'85 formats). Columns 3 and 4 of Table 6 show the number of nodes #N in ROBDD and WDD. Columns 5

³ <http://akebono.stanford.edu/users/luca/doc/cuddIntro.html>.1998. *Somenzi F.* CUDD: CU Decision Diagram Package. University of Colorado at Boulder.

⁴ <http://www.rs.e-technik.th-darmstadt.de/~sth>. 2000. *Höreth S., Blank C.* TUD Decision Diagram Package. Darmstadt University of Technology. Germany.

Table 6. Size (number of nodes) in ROBDD, WDD, and LDD for test specimens LGSynth'93

TEST	I/O	ROBDD WDD		LDD (of EDIF)			LDD (of ISCAS'85)		
		#N	#N	#L	#N	CPU	#L	#N	CPU
c432	36/7	1064	1209	21	455	0.01	17	336	<0.00
c499	41/32	25 866	29 561	23	889	0.03	11	365	0.01
c880	60/26	4053	4048	35	604	0.03	24	605	0.01
c1355	41/32	29 561	29 651	23	929	0.04	24	993	0.04
c1908	33/25	5526	5944	51	921	0.04	40	1464	0.07
c2670	233/140	1850	3939	40	1374	0.10	32	2026	0.14
c3540	50/22	23 828	—	62	1902	0.11	47	2760	0.22
c5315	178/123	1719	2504	68	3192	0.31	49	4156	0.47
c6288	32/32	—	—	124	4320	0.38	124	4318	0.23
c7552	207/108	2213	—	64	4627	0.49	43	6059	0.99

Table 7. Size of LDD for sequential circuits LGSynth'93

TEST	#I/O	#G	#NR	#L	#N	MEM	t
s208.1	11/1	136	672	58	214	3550	0.00
s344	10/42	232	1640	90	351	5848	0.01
s349	10/42	236	896	91	358	5941	0.02
s382	4/6	253	1403	129	416	7510	0.03
s400	4/6	267	1423	134	440	7957	0.01
s526n	4/6	272	4314	73	536	7344	0.02
s420.1	19/1	278	2600	114	7248	6312	0.01
s444	4/6	295	1450	80	476	6829	0.01
s526	4/6	298	2187	83	552	7755	0.03
s641	36/27	359	4828	101	515	7709	0.01
s713	36/27	397	4727	107	591	8615	0.01
s838	36/2	589	5092	225	898	14 595	0.06
s838.1	35/1	562	9307	226	916	12 764	0.06
s1196	15/14	705	11 556	34	1175	21 224	0.04
s1423	18/5	870	92 049	280	1388	21 317	0.08

and 6 show the number of levels #L, total number of nodes #N and time of CPU in seconds (t) required to construct an LDD set for representing the circuit. Experimental data in EDIF format were transformed to BLIF format and then the CUDD package was applied to design a ROBDD. Experimental data show that ROBDD contains far more number #NR of nodes than LDD #N.

It is not a simple matter to construct ROBDD and WDD for a 32-bit multiplier (c6288), whereas an LDD for this circuit can be generated in reasonable time.

Table 7 shows the results obtained in generation of LDD for sequential circuits. The memory size for storing synthesized LDD was compared with the memory size for representing other diagrams, in particular, ROBDD. For this purpose, data in different formats were used.

7. FUNCTIONAL MODELING ON LDDS

Computation of functions with word-level models are believed to be much more effective in speed than computation at bit level since several bits are computed in parallel. Moreover, computations with linear polynomials are faster than with nonlinear polynomials due to simplification. In this

Table 8. An algorithm for computing Boolean functions with LDD models

```

procedure Evaluate(circuit, variables_assignment):
{ buildLDD(circuit)
  for ( $\forall$  level l in the network)
  { for ( $\forall$  output t of the level l)
    for ( $\forall$  terminal node i)
      { tmp  $\leftarrow$  tmp +  $a_{i,t}$ ;
        tmp  $\leftarrow$  tmp +  $a_{0,t}$ ;
         $y_g = \Xi^{J_t}\{tmp\}$ ;
        Extract LSB or MSB from tmp;
        for ( $\forall$  gates destined from the gate g)
          SendMail(#tree, #node,  $y_g$ ); } } }

```

Table 9. Comparison of the time required to compute the Boolean function realized by the circuit via simulation and from an LDD model

TEST	IN	#L	<i>t</i> [ms]	<i>t1</i> [ms]	TEST	IN	#L	<i>t</i> [ms]	<i>t1</i> [ms]
C17	5	6	0.002	0.004	C2670	233	45	0.379	0.491
C432	36	28	0.109	0.150	C3540	50	65	0.447	0.593
C499	41	25	0.148	0.168	C5315	178	78	1.240	1.580
C880	60	39	0.188	0.339	C6288	32	245	1.100	2.194
C1355	41	43	0.241	0.288	C7552	207	66	2.011	2.340
C1908	33	54	0.247	0.252					

section, we study modeling (simulation) of functional outputs of a circuit for given inputs in a *LinearDD* system.

Let us consider a multi-level combination circuit without feedback. An algorithm for computing the functions realized by a circuit based on an LDD model is shown in Table 8. In the *LinearDD* system, computed functions are represented by two classes. The first class, *LinearDecisionDiagram*, contains a *Compute* procedure, which reads LDD inputs and computes LDD outputs. The second class, *Mask*, realizes a masking operator and contains information on LDD fan-out. The *SendValueToFanOut* function hides the respective LDD outputs and transfers their values to the nodes whose inputs are connected to the LDD outputs.

Linear decision diagrams for test circuits based on LGSynth'93 databases were generated in experiments. Computation time was measured in seconds for PC Pentium III 450 MHz with 128 Mb RAM. Table 9 shows the mean CPU time for computing the outputs of a circuit in EDIF (*t*) and LDD (*t1*) formats for 10000 randomly generated input patterns. In the first case, output values were computed directly from the circuit structural description using logical operations according to the realized function (digit by digit)—the most effective circuit simulation technique. According to our experiments, time *t* is comparable with time *t1*.

8. ESTIMATION OF NETLISTS FROM AN LDD SET

Initial data for the algorithm described in Table 10 is the *l*th LDD in an *m*-LDD system, which has *n* nonterminal nodes. The result contains the type of an element and its inputs and outputs. In the course of analysis, the *l*th LDD ($l \in 1, \dots, m$) and fan-out of elements of the *l*th level are stored in masking operators, parameters *#tree*, and *#node*.

Table 10. Algorithm for estimating netlists from an LDD model

```

procedure NetlistExtracting(set of LDDs):
{ for( $\forall$  LDD  $l$  in the set of LDDs)
  { for( $\forall$  exponent  $t$  of the weight  $2^{J_t}$ )
    { for( $\forall$  terminal node  $i$ )
      {  $n_t \leftarrow$  number of terminal elements  $a_{i,t}$  different from zero;
         $n_{t-} \leftarrow$  number of terminal elements  $a_{i,t}$  lower from zero;
         $j_t \leftarrow \lceil \log_2 n_t \rceil + 1$ 
         $h \leftarrow$  exponent of  $\Xi^h$  corresponding to the weight  $2^{J_t}$ ,
        If  $h = t$  and  $a_{0,t} = 0$  add an  $n_t$ -input EXOR gate,
        Else if  $h = t$  and  $a_{0,t} \neq 0$  add an  $n_t$  input EXNOR gate,
        Else if  $h \neq t$  and  $a_{0,t} = 2^{j_t-1} - n_t + n_{t-}$  add an  $n_t$  input AND gate,
        Else if  $h \neq t$  and  $a_{0,t} = 2^{j_t-1} - 1 + n_{t-}$  add an  $n_t$  input OR gate,
        Copy outputs from the masking operators.
        Complement all the inputs which corresponds to the terminal elements
        lower than zero. }}}}

```

Masking operators are not used to transform a circuit from LDD format to a netlist. Netlists are generated from a system of diagrams in two steps.

(1) Transformation of a given set of linear decision diagrams to internal format, which stores information on elements and their inputs and outputs by the algorithm described above.

(2) Transformation of the internal format in one of standard circuit representation forms (EDIF, XNF) or hardware description languages Verilog and VHDL.

Transformation results of EDIF to LDD and LDD to a netlist in EDIF and XNF formats are given in [18]. Verilog and VHDL structures can also be generated from the LDD internal format. Details and specifications for these structures in hardware description languages are given in [19].

9. EXTENSION OF A SYSTEM

In this section, we describe a method of extending the *LinearDD* system, viz., extension of the library of primitives for single-electron transistor nanotechnologies (Table 2). These technologies are used for producing devices realizing arrays containing demultiplexer-type switches at nodes. Switches can be realized on a single-electron transistor or a *quantum-dot* cell [11]. Both technologies are based on the Josephson contact effect and believed to be feasible due to the advances in the technology of superconductive materials for creating devices on Josephson contacts [12] and may attain a clock rate of tetraHerz for nanoscale sizes.

Hypercube data structures. Representation of Boolean functions as cubes and hypercubes, and the solutions of optimization problems based on them are widely used in the classical logical design. Another field of application for hypercubes is communication problems, in particular, design of multi-processor systems. CAD-systems for logical circuit design in a three-dimensional space must be based on the principles of three-dimensional realization of a logical function with a library of primitives. In this case, the hypercube acts as a carrier for the functional properties of a logical circuit and topology of its physical realization [20]. This and other specific features of nanotechnologies [11–23] underlie the design of primitives with \mathcal{N} -hypercubes [20].

\mathcal{N} -hypercube. The functions of the *LinearDD* system are extended by imbedding the total solution tree into an \mathcal{N} -hypercube structure according to the scheme

$$\text{Function (circuit)} \iff \text{Total solution tree} \iff \mathcal{N}\text{-hypercube.}$$

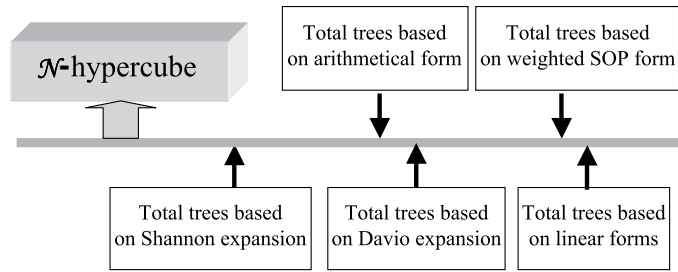


Fig. 5. Library of 3D-primitives in the *LinearDD* system based on imbedding of a total solution tree into an \mathcal{N} -hypercube.

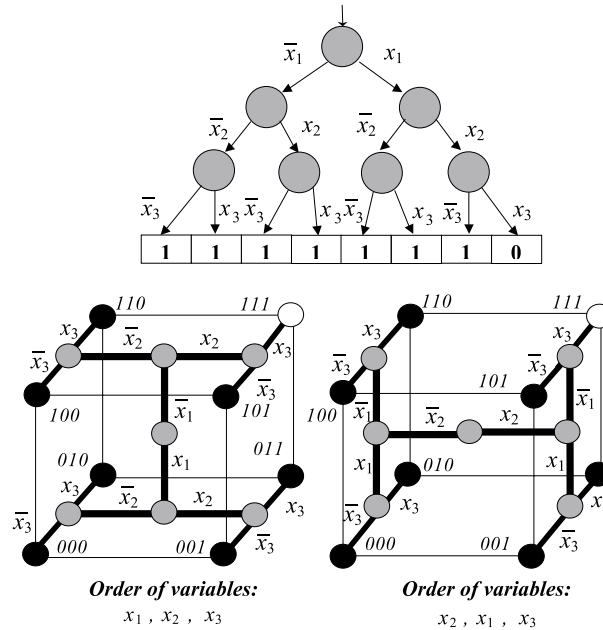


Fig. 6. Imbedding of a tree into an \mathcal{N} -hypercube: the \mathcal{N} -hypercube structure depends on the order of variables in the initial total solution tree.

Given two graphs $G = (V, E)$ and $H = (W, F)$, let us *imbed* the graph G into the graph H . The imbedding procedure realizes a Boolean function as a hypercube. Information in the resultant \mathcal{N} -hypercube is subdivided according to the new structural properties of the cube and the type of the imbedded tree. Some decomposition (Shannon and Davio decompositions, arithmetical analog of the Davio decomposition, and decomposition for word-level forms, including linear forms (Fig. 5)) of a Boolean function by one variable is realized at the node of the solution tree.

Example 5. Figure 6 shows the imbedding of a total solution tree of a Boolean function of three variables into an \mathcal{N} -hypercube by different methods. The values of the function from intermediate nodes realizing the decomposition of the function by variables for a given order of variables are transferred to the node of the \mathcal{N} -hypercube.

Imbedding of an LDD into \mathcal{N} -hypercube. Examples of linear decision diagrams imbedded into 1D-, 2D-, and 3D-hypercubes are shown in Fig. 7. The \mathcal{N} -hypercubes of many-input circuit elements and, in general, levels are designed by subcube build-up procedures. The extended *LinearDD* system contains libraries of 2D and 3D primitives of two- and three-input elements, which are oriented for nanotechnologies, in particular, single-electron technology [11].

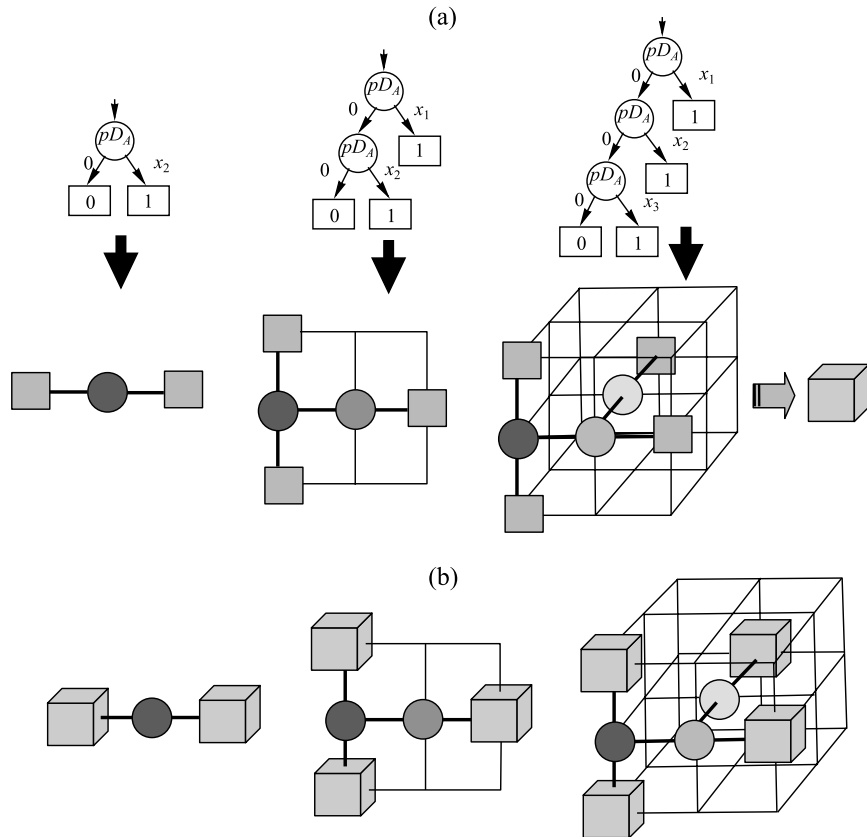


Fig. 7. Design of 3D models of primitives based on \mathcal{N} -hypercubes: (a) LDD model for 1-, 2-, and 3-input AND elements and the corresponding 1D, 2D and 3D \mathcal{N} -hypercubes; (b) 4D, 5D, and 6D \mathcal{N} -hypercubes of 4-, 5-, and 6-input levels of the circuit.

Neural networks, evolution algorithms, and information measurement methods are used in the artificial intelligence modules of the extended *LinearDD* system (Table 2) [24, 25].

10. CONCLUSIONS

Circuit functional description must be linearized for compacting the representation of large circuits, as demonstrated by experiments with our *LinearDD* system having structural and functional flexibility and compatibility with lower-level tools. For example, the interface with EDIF and XNF formats in the *LinearDD* system provides the “transparency” of linear decision diagrams for forming netlists. In particular, XNF-netlist can be generated from linear decision diagrams and then the Xilinx tools can be applied to design an FPGA prototype suitable for physical mapping. LDD may be used as an intermediate format for generating netlists from the description in HDL, or transform the circuit model into a description in HDL for further synthesis or verification of ASIC or FPGA. Experimental results demonstrate that any network of large logical (industrial) elements can be transformed into linear decision diagrams.

The rationality of the architecture of the *LinearDD* system is corroborated by the possibilities for its extension, including for use in future technologies. The system, for instance, contains a library of three-dimensional primitives for logical design of circuits by nanotechnologies.

REFERENCES

1. Akers, S., Binary Decision Diagrams, *IEEE Trans. Comput.*, 1978, vol. 27, no. 6, pp. 509–516.
2. Bryant, R.E., Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Comput.*, 1986, vol. 35, no. 6, pp. 677–691.
3. Rudell, R., Dynamic Variable Ordering for Ordered Binary Decision Diagrams, *Proc. Int. Conf. Comput. Aided Design*, 1993, pp. 42–47.
4. Bryant, R.E. and Chen, Y.A., Verification of Arithmetic Functions with Binary Moment Diagrams, *Proc. Design Automat. Conf.*, 1995, pp. 535–541.
5. Höreth, S. and Drechsler, R., Formal Verification of Word-Level Specifications, *Design Automat. and Test Conf.*, Germany, 1999, pp. 52–58.
6. Malyugin, V.D., Realization of Corteges of Boolean Functions by Linear Arithmetical Polynomials, *Autom. Telemekh.*, 1984, no. 2, pp. 114–121.
7. Yanushkevich, S.N., Shmerko, V.P., Malyugin, V., and Dziurzanski, P., Linearity of Word-Level Circuit Models: A New Understanding, *Proc. 11th IEEE/ACM Int. Workshop on Logic & Synthesis*, New Orleans, 2002, pp. 67–72.
8. Shmerko, V.P., Malyugin's Theorems: A New Concept in Logical Control, VLSI Design, and Data Structures for New Technologies, *Autom. Telemekh.*, 2004, no. 6.
9. Dziurzanski, P., Malyugin, V.D., Shmerko, V.O., and Yanushkevich, S.N., Linear Models for Circuits on Many-Valued Elements, *Autom. Telemekh.*, 2002, no. 6, pp. 99–119.
10. Yanushkevich, S.N., Dziurzanski, P., and Shmerko, V.P., The Word-Level Models for Efficient Computation of Multiple-Valued Functions, Part 1: LAR Based Model, *Proc. IEEE Int. Symp. Multiple-Valued Logic*, Boston, 2002, pp. 202–208.
11. Asahi, N., Akazawa, M., and Amemiya, Y., Single-Electron Logic Device Based on the Binary Decision Diagram, *IEEE Trans. Electron. Devices*, 1997, vol. 44, no. 7, pp. 1109–1116.
12. Likharev, K. and Semenov, V., RSFQ Logic/Memory Family: A New Josephson Junction Technology for Sub-Terahertz Clock Frequency Digital Systems, *IEEE Trans. Appl. Supercond.*, 1991, vol. 1, pp. 3–28.
13. Muroga, S. and Ibaraki, T., Design of Optimal Switching Networks by Integer Programming, *IEEE Trans. Comput.*, 1972, vol. 21, pp. 573–582.
14. Papaioannou, S.G., Optimal Test Generation in Combinational Networks by Pseudo-Boolean Programming, *IEEE Trans. Comput.*, 1977, vol. 26, pp. 553–560.
15. Minato, S., *Binary Decision Diagrams and Applications for VLSI CAD*, New York: Kluwer Academic, 1996.
16. Kukharev, G.A., Shmerko, V.P., and Yanushkevich, S.N., *Tekhnika parallel'noi obraborki binarnykh dannykh na SBIS* (Parallel Processing of Binary Data on VSLI), Minsk: Vysshaya Shkola, 1991.
17. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *Introduction to Algorithms*, Cambridge: MIT Press, 2001.
18. Sizeland, R. and Xie, Y., Linear Decision Diagrams Based Mapping for Rapid Prototyping, *Autom. Telemekh.*, 2004, no. 6.
19. Lu, D.C., Rahman, C., and Wahid, K., VHDL Constructs in Linear Word-Level Decision Diagram Based Specification, *Autom. Telemekh.*, 2004, no. 6.
20. Yanushkevich, S. and Shmerko, V., *Modelling of Logic Primitives in a Nanospace*, Technical Report, Dept. of Electrical and Comput. Eng., Univ. of Calgary, Canada, 2003.
21. Derycke, V., Martel, R., Appenzeller, J., and Avouris, P., Carbon Nanotube Inter- and Intra-molecular Logic Gates, *Nano Lett.*, 2001, vol. 1, pp. 453–456.

22. Ellenbogen, J.C. and Love, J.C., Architectures for Molecular Electronic Computers: Logic Structures and an Adder Designed from Molecular Electronic Diodes, *Proc. IEEE*, 2000, vol. 88, no. 3, pp. 386–426.
23. Huang Yu, Duan, X., Cui Yi, Lauhon, L.J., Kim, K.-H., and Liber, C.M., Logic Gates and Computation from Assembled Nanowire Building Blocks, *Science*, 2001, vol. 294, pp. 1313–1317.
24. Luba, T., Moraga, C., Yanushkevich, S., Shmerko, V., and Kolodziejczyk, J., Application of Design Style in Evolutionary Multi-Level Networks Synthesis, *Proc. IEEE Sympos. Digital Syst. Design*, Maastricht, Netherlands, 2000, pp. 156–163.
25. Cheushev, V., Yanushkevich, S., Moraga, C., and Shmerko, V., *Flexibility in Logic Design: An Approach Based on Information Theory Methods*, Research Report, Forschungsbericht 741, Univ. of Dortmund, Germany, 2000.

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