



Three-Dimensional Feedforward Neural Networks and Their Realization by Nano-Devices

VLAD P. SHMERKO and SVETLANA N. YANUSHKEVICH

Department of Electrical and Computer Engineering, University of Calgary (E-mail: {shmerko;yanush}@enel.ucalgary.ca)

Abstract. The three-dimensional (3D) model of a feedforward neural network (NN) based on so called \mathcal{N} -hypercube topology is proposed. The \mathcal{N} -hypercube is different from the classical hypercube used in communication theory, and in Boolean algebra. This new structure has been created based on a novel algorithm for embedding a binary decision tree and binary decision diagram into a \mathcal{N} -hypercube. It is shown that \mathcal{N} -hypercube topology is a reasonable solution to implement NN of threshold gates, in particular, on the single-electron devices. The 3D design methodology of feedforward NN is oriented to technology mapping to nanodevices. Results of extensive experimental study of feedforward networks consisting of over 3500 \mathcal{N} -hypercubes are presented.

Keywords: decision tree, hypercube, single-electron logic, three-dimensional neural network

1. Introduction

It has been justified in a number of papers that 3D topologies of computer elements are optimal, physically realistic models for scalable computers (Frank et al. 1998). Nanoelectronic devices are expected to be 3D. This has motivated search for new design methodologies to meet expected computing demands in the nanometric era. This paper contributes in design methodologies of such tools, namely, we focus on feedforward NNs.

One of the primary problems in the development of large nanoICs circuits is the lack of appropriate design methodologies that effectively utilize CAD tools.

Recent results indicate that the most powerful NNs have 10^2 – 10^3 inputs and about 10^3 – 10^4 neuron-like gates of 10^2 – 10^3 inputs each (Fiesler and Beale (eds.) 1997). It corresponds to about 10^8 connections and processing speed about 10^{11} “connections per second”. These results are based on the state-of-the-art technology and 2D design methodology. It should be noted that biological NN realize 10^{14} connections with a speed of 10^{16} connections per second in 3D space. It is expected that only breakthrough can change such drastic limitations. This is the motivation to develop a method of mapping NN into 3D space.

We study a computational framework for models of a linear threshold neuron in nanometric space. The crucial idea is to utilize a hypercube topological data structure that has been well studied in communication, supercomputer design, and logic circuit synthesis. However, direct application of the traditional hypercube model to 3D representation of a set of simple computational nanoelements leads to serious difficulties. We propose to utilize the unique properties of a hypercube topology by modifying the hypercube structure. It is based on the similarities between the topological structure of binary decision trees and binary decision diagrams, and of Single Electron Transistor (SET).

The problem of mapping 2D models of SET into 3D space corresponds to the problem of embedding a binary decision tree or binary decision diagram into appropriate 3D model. This problem is only solved, for simple forms of decision trees. Binary decision trees are relevant to functional formulation, and thus, are very different from conventional hypercubes used in communication and supercomputer design. Hence, it is critical to modify the hypercube in order to embed binary decision trees and binary decision diagrams into 3D hypercube space.

The main feature of the methodology described in this paper is the application to SET technology. The second important feature is the development of a library of 3D threshold elements and 3D macro components, allowing the design of nanocircuits of arbitrary complexity. Finally, the proposed methodology is generic because it does not limit the types of decision trees, so it is possible to design technology independent 3D structures.

In this paper we develop a new data structure, \mathcal{N} -hypercube, based on the modified classical hypercube. The differences between the classical hypercube and the proposed \mathcal{N} -hypercube is in functions of the nodes. However, there are many common features: classical hypercube and \mathcal{N} -hypercube have the same rules for encoding the terminal nodes (Gray code) and the same measurements, except special evaluations of the effectiveness of a resulting 3D architecture.

It is worthy to note that the hypercube data structure together with the Reduced Ordered Binary Decision Diagram (ROBDD) are the state-of-the-art tools of advanced logic circuit design. The spatial topology of the hypercube is suitable for mapping network models into 3D space, and, thus, is an appropriate model for nanotechnologies.

We refer to the most important recently reported results as follows.

- (a) The concept of nanocomputing based on Josephson junction technology is discussed in (Likharev 1999; Takahashi et al. 2000; Wasshuber 2001). In Takahashi et al. (2000) applications of SET to neuron modeling are reported.

- (b) Mapping a ROBDD into a hexagonal nanowire network controlled by Schottky wrap gates have been introduced in Asahi et al. (1997) and Yamada et al. (2001). In our paper we utilize and generalize the unique correspondence of SET topology and ROBDD topology.
- (c) Survey on NN hardware implementation and its trends is given in Fiesler and Beale (eds.) (1997) and Webster (1999).
- (d) Boolean functions computing by feedforward NN is introduced in Siu et al. (1991). In our approach, we study the 3D feedforward network (without feedback) that consists of linear threshold gates.
- (e) Topological properties of hypercubes have been introduced in Saad et al. (1988). In Roth (1999), Boolean function manipulation based on hypercubes is discussed. We extend the classical hypercube topological data structure toward the \mathcal{N} -hypercube.

The results of our study are introduced as follows. In the next section we give the basic definitions starting with graph theory and classical hypercube. Embedding binary tree into hypercube is discussed in section 3. In section 4 a new hypercube data structure, \mathcal{N} -hypercube, is introduced. The method of 3D NN design via embedding a decision tree into a \mathcal{N} -hypercube is discussed in section 5. We show in section 6 that \mathcal{N} -hypercube is a reasonable model for a SET technology. We focus on library of nanoneurons, and introduce our approach to 3D NN design in section 7. The results of experimental studies are presented in section 8.

2. Basic Definitions

The theoretical basis of our study combines

- (i) NN design methodology, that focus on feedforward threshold network,
- (ii) Graph data structures, including decision trees, decision diagrams, and hypercubes, and
- (iii) SET nano-technology.

Definition 1. *The threshold gate is a computational element that calculates the sum of n inputs $x_i \in (0, 1)$ weighted with weights w_i . The sum is converted into digital output y by comparing the sum with a given threshold θ*

$$y = f\left(\sum_{i=1}^n w_i x_i - \theta\right) = \begin{cases} 1, & \text{if } \sum_{i=1}^n w_i x_i \geq \theta, \\ 0, & \text{if } \sum_{i=1}^n w_i x_i < \theta. \end{cases}$$

Adapting the weights and threshold, a linearly threshold gate computes any linear separable Boolean function of n inputs. The feedforward network is defined as a network of interconnected linear threshold gates, and NN to

be a feedforward network of these gates. The tree is an appropriate model to describe a set of threshold gates.

Definition 2. *A tree is a rooted acyclic graph in which every node but the root has indegree 1.*

The form of representation of a tree is critical to the design methodology that is based on principle of embedding of the tree into a hypercube. In a tree, for any vertex v there exists a unique path from the root to v . The length of this path is called the *depth* or *level* of v . In logic design, the path corresponds to sum-of-product (SOP) terms of the Boolean function. The height of a tree is equal to the largest *depth* in the tree. A node with no children is a *terminal (external) node* or *leaf*. Terminal nodes carries information such as, for example, the function value (in ROBDD). A nonleaf node is called an *internal node*. The function of a node can be given in abstract form (communication problems) or specified exactly (logic design problems).

This edge model is suitable, to example, to describe transmission of data, so that data are sent in only one direction at a time, up or down. The *size* of the tree is the number of nodes and it often measures of software or hardware requirements.

If all nodes are grouped with the same depth together, the network is to be arranged in *layers*. Hence, the depth of the network can be interpreted as time of parallel computations.

Example 1. *A feedforward network is defined as an acyclic labeled directed graph with distinguished input nodes of indegree zero, internal nodes of arbitrary indegree, and distinguished output nodes.*

Definition 3. *A complete n -level binary tree is the tree with 2^k nodes on level k for $k = 0, \dots, n - 1$.*

The 2^n -leaf complete binary tree has a level hierarchy (levels $0, 1, \dots, n$). The root is associated with level zero and its two children are on level one, and the j -th node on level i is denoted by (i, j) for $0 \leq j < 2^i$. In the complete binary tree, each node is either a leaf or has degree 2. The complete binary 3-level ($n = 3$) tree is given in Figure 1.

Definition 4. *An Ordered Binary Decision Diagram (OBDD) for Boolean function f with respect to ordering π is a directed acyclic graph consisting of non-terminal nodes labelled with the variables of f and terminal nodes labeled with 0 and 1.*

OBDD has a number of useful properties. In particular, an arbitrary Boolean function can be represented by an OBDD, and using reduction rules, the OBDD can be transformed into a canonical form of the Boolean function, i.e., ROBDD.

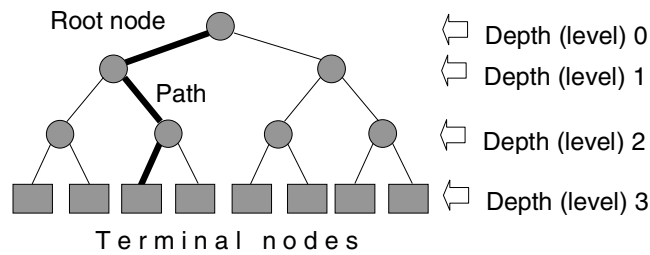


Figure 1. Complete binary tree.

Definition 5. *Hypercube is a graph data structure that consists of d dimensions and 2^d vertices, encoded by $(g_0 \dots g_{d-1})$. Two vertices $(g_0 \dots g_{d-1})$ and $(g'_0 \dots g'_{d-1})$ are connected by an edge if and only if they differ in exactly one bit (Gray code). There are $d \times 2^{d-1}$ edges in a d -dimensional hypercube.*

The *fan-out* (i.e., degree) of every node is n , and the total number of communication links is $\frac{1}{2}N \log_2 N$. Each node of the n -dimensional hypercube can be specified by the binary address $(g_{n-1}, g_{n-2}, \dots, g_0)$, where bit g_i corresponds to the i -th dimension in a Boolean space. The dimensions are specified by the set $\{0, 1, \dots, n-1\}$.

3. Embedding a Binary Tree into a Hypercube

The crucial phase to design a hypercube space topology is a mapping of a 2D graph into a hypercube. This procedure is called *embedding* of a guest graph into a hypercube (host graph).

3.1. Forms of a binary tree representation

Definition 6. *An embedding, $\langle \varphi, \alpha \rangle$, of a guest graph G into a host graph H is the one-to-one mapping $\varphi: V(G) \rightarrow V(H)$, along with the mapping α that maps an edge $(u, v) \in E(G)$ to a path between $\varphi(u)$ and $\varphi(v)$ in H . The embedding of a graph G into a graph H is an injection (one-to-one mapping) of the nodes in G to the nodes in H .*

For simplification but without loss of generality, we will discuss embedding a complete binary tree into a hypercube. The form of representation of the binary tree (specification) and conditions of embedding are critical to choosing the method of embedding.

Definition 7. *Node specification of an arbitrary binary tree includes a function of the node (abstract machine that computes the outputs accordingly to some algorithm) and its topology information (root, internal node,*

terminal node, coordinates). **Edge specification** includes a function of the edge (abstract data or specified value transmission) and its topology information (indexes of related nodes). Node and edge specifications have different additional **attributes** such as the level number, the path number, and the value of terminal node.

The combination of a node and edge specifications together with attributes allows generating different forms of a binary tree representation. We distinguish three forms of the binary tree representation.

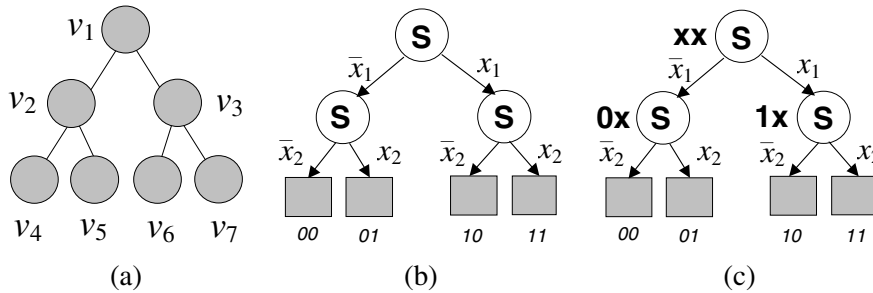


Figure 2. The forms of representation of a binary tree: Specification 1 is typical for communication problems (a), Specification 2 is used for Boolean function manipulation and decision diagram design (b), and Specification 3 is used for representation of a decision space in 3D space (c).

Specification-1: The binary tree is a particular network architecture, that simply carries information about dual connections of each node (Figure 2a). This formulation is typical for many communication problems.

Specification-2: The binary tree carries information about functionality of the logic circuit (Figure 2b). A node of the binary tree is associated with Shannon expansion with respect to a variable (denoted by S). Algebra of relationships of nodes (variables) is known as a cube algebra to manipulate Boolean functions. In this formulation, a number of optimization problems is solved in terms of logic circuit design. Simplification of this tree based on reduction rules gives a ROBDD.

Specification-3: The binary tree carries information about functionality of the logic circuit and, in addition to Specification-2, the topology (coordinate). The nodes of the binary tree are associated with Shannon expansion with respect to each variable and coordinate in 3D space (Figure 2c).

The algorithms of embedding a complete binary tree given in Specification-1 and Specification-2 are introduced in many public-

ations, in particular, (Saad et al. 1988; Öhring et al. 1995). We are focused on the unsolved problem of embedding the tree given in Specification-3 into a hypercube.

3.2. Specification-1

Let the embedding conditions be formulated as follows: (i) each node of a tree is labeled by v_i , and (ii) the number of nodes in a tree is equal to or less than the number of nodes in the hypercube. Even in the simplest form of a binary tree description, Specification-1, the effectiveness of the embedding strongly depends on a number of parameters. The *dilation cost* of an embedding of the graph G in the hypercube H is the maximum distance in H between any two neighbor nodes in G . This cost gives a measure of the proximity in H of the neighbor nodes in G under the embedding. The *expansion* is the ratio $|V(H)|/|V(G)|$. The *load* of the embedding is the maximum, over all host nodes, of the number of guest nodes mapping into it. The *edge-load* of edge $e \in E_H$ in the embedding is the maximum number of edges of G that are routed by g over the edge e . The *edge-congestion* is the maximum edge load, over all host edges.

In our design methodology and experimental study we use these parameters to measure the designed 3D structures. In addition, we have found that 3D Voronoi diagrams are very useful tools for measuring in 3D space (Okabe et al. 1992).

In terms of Specification-1, the task is formulated as finding a balance between edge load and communication time. It requires to minimize such cost parameters as dilation, expansion, and edge-congestion. For example, there exists an optimal embedding of a complete binary tree with load and expansion 1, and dilation and edge-congestion 2 into a hypercube (Öhring et al. 1995).

Hence, in Specification-1 functions of nodes are different. Their functions are not important, and only edges between the nodes carry useful information. The task of embedding is formulated in terms of edges, i.e. distance, dilation cost, etc.

3.3. Specification-2

The idea of a hypercube in a logic circuit design is that an n -bit string can be visualized geometrically, as a vertex of a hypercube that consists of 2^n vertices, each of which is labelled with an n -bit string (Roth 1999). 0D hypercube is a point, a 1D hypercube is a line, a 2D hypercube is a square, and 3D hypercube is a cube. The truth table of an n -input Boolean function can be represented in an n -dimensional Boolean space. There is one axis for each

variable which can take two values: 0 and 1. Each node is labeled with its coordinate in the n -dimensional space. Specification-2 is utilized in the minimization a Boolean functions.

The minterm of an n -variable Boolean function can be represented by an n -bit integer (the minterm number) where each bit indicates whether the corresponding bit variable is complemented or uncomplemented. To deal with product terms that are not minterms, where some variables do not appear at all, we must represent three possibilities for each variable in a general product term: 1 – uncomplemented, 0 – complemented, and \mathbf{x} – does not appear. For example, given a function of six variables, $x_1, x_2, x_3, x_4, x_5, x_6$, we write the following product terms and their cube representations: $x_1, x_2, \bar{x}_3 = 110\mathbf{xxx}$, $\bar{x}_1 x_5 x_6 = 0\mathbf{xxx}11$ (the hypercube notation is conventional in logic design).

A node in the binary decision tree realizes the Shannon decomposition

$$f = \bar{x}_i f_0 \oplus x_i f_1, \quad (1)$$

where $f_0 = f(x_i = 0)$, and $f_1 = f(x_i = 1)$, for all variables in f .

Consequently, the embedding conditions can be formulated as follows:

- (i) Each node realizes Shannon expansion (1),
- (ii) The nodes are distributed over levels,
- (iii) Each level, except level 0, corresponds to a variable x_i ,
- (iv) Each path from the root node to a terminal nodes corresponds to a minterm in SOP representation, determined as the product of labels at the edges, and
- (v) A terminal node is labelled with 1 if the path from the root node to the terminal node (minterm) is equal to 1.

Note, that since the variables appear in a fixed order, such tree is an *ordered* binary decision tree.

Example 2. Five minterms of the Boolean function $f = x_1 \bar{x}_2 \vee \bar{x}_3$ given by the truth table (Figure 3a) are represented by the tree with five active terminal nodes (the value 1) (Figure 3b). The cube representation (Figure 3c) implies representation of the function in cube terms: $\langle \mathbf{xx}0, 01\mathbf{x} \rangle$.

It follows from the above description of input data that, in addition to the Specification-1 of a tree, levels become the carriers of information. Based on this extension, effective methods to manipulate edges have been developed with the goal to minimize the number of hypercubes to represent a given Boolean function.

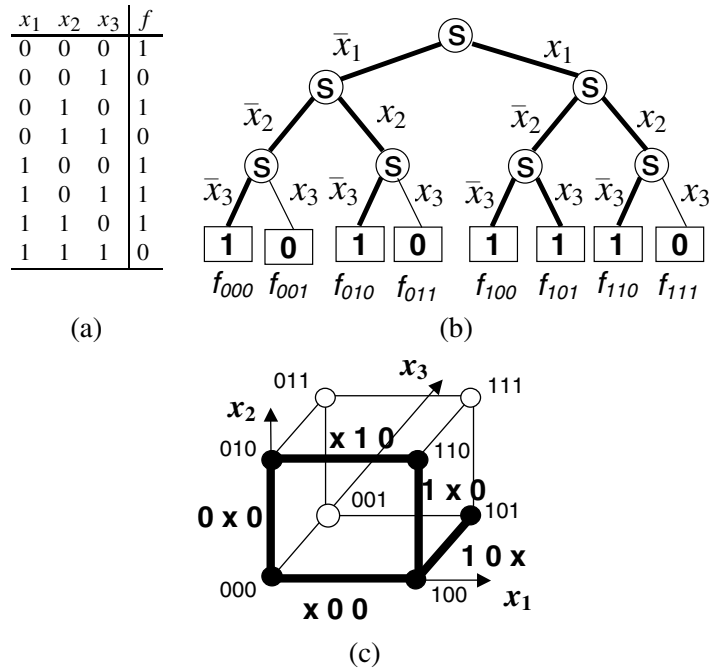


Figure 3. Truth table of the function $f = x_1\bar{x}_2 \vee \bar{x}_3$ (a), the binary decision tree (b), and hypercube representation (c).

3.4. Specification-3

The complete binary decision tree in Specification-3 is given in Figure 4. We observe that this tree is different from Specification-2 by labels of non-terminal nodes. The embedding conditions for this tree are the same as for Specification-2 except that each node is labelled with a coordinate.

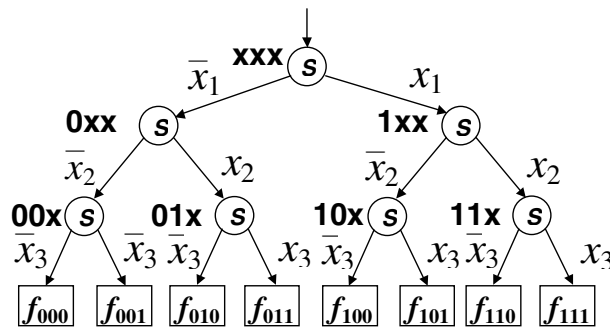


Figure 4. A complete binary decision tree to represent an arbitrary Boolean function of three variables; each node is labelled with a distinguished coordinate.

4. \mathcal{N} -hypercube

As mentioned above, there is a number of reasons to develop a new data structure based on hypercube topology. The main reason is to overcome limitations of the hypercube topology while embedding a binary decision tree. Specification-3 is of practical interest because it satisfies technological requirements of nanodevice fabrication. In this section, we focus on modification of a classical hypercube with the purpose to solve the formulated problem.

4.1. Topology

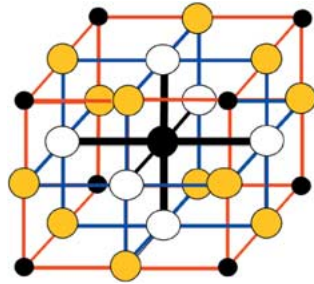


Figure 5. The complete \mathcal{N} -hypercube.

The \mathcal{N} -hypercube topology is different from the topology of a classical hypercube, in the number of nodes, types of nodes, and connections (Figure 5).

Number of nodes. The classical hypercube contains 2^n nodes, $n = 0, 1, \dots, M$, while the \mathcal{N} -hypercube includes $\sum_{i=0}^n 2^{n-i} C_i^n$ extra nodes, so it is possible to manipulate $2^n + \sum_{i=0}^n 2^{n-i} C_i^n$ nodes while designing the 3D circuit.

Type of nodes. In the classical hypercube, the nodes are not differentiated by their type, i.e. each node can be a generator or receiver of information. In contrast, in \mathcal{N} -hypercube there are three types of nodes, and their functions are different.

Connections. In traditional design methodology, hypercubes can communicate via appropriate nodes accordingly to a given strategy, i.e. there are no special requirements to the nodes. Communication in the \mathcal{N} -hypercubes is scheduled by intermediate nodes.

It should be noted that the connection strategy is closely related to the well-studied problem of the ROBDD design (Bryant and Meinel 2002).

In Figure 6 various types of \mathcal{N} -hypercube nodes are illustrated.

Example 3. Let $n = 3$. The 3D hypercube consists of 8 nodes and \mathcal{N} -hypercube involves $\sum_{i=0}^3 2^{3-i} C_i^3 = 27$ nodes: 8 terminal nodes, 12 intermediate nodes, 6 face nodes and one root node.

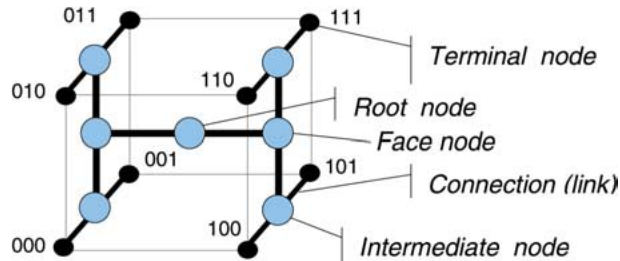


Figure 6. \mathcal{N} -hypercube topological structure consists of four types of nodes: terminal nodes, intermediate nodes, face node, and the root node.

In addition, we note that \mathcal{N} -hypercube can be considered as technology dependent model because of its close relationship to the SET topology.

4.2. \mathcal{N} -hypercube as technology dependent data structure

Consider two simplest topological structures: a node of a binary decision tree and SET (Figure 7). In the binary tree, a messenger electron travels along the path and reaches 1 or 0 terminal (Asahi et al. 1997). A value of the Boolean function is recognized by observing which terminal the electron reaches. The Shannon expansion of a node is implemented as two-way clocked and switching controlled variable x . Physically, it consists of four tunnel junctions. A messenger electron travel through the entry branch to the exit branch that corresponds to the binary value of the input signal: if input signal x is a positive (1), then the electron is transported to 1-branch, otherwise, 0-branch.

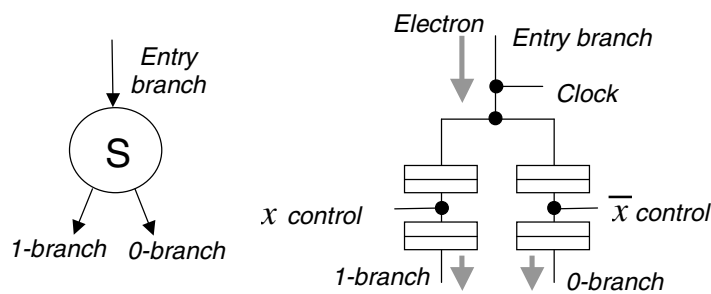


Figure 7. The binary decision tree (a) and a model of a SET (b).

The set of two SETs corresponds to a binary decision tree with four nodes that can be embedded into a 2D hypercube. If the set contains three SETs, its 2D model is a tree with eight nodes that is embedded into a 3D \mathcal{N} -hypercube. To design a 4D \mathcal{N} -hypercube from the binary tree with 16 nodes, we need to add new links to connect intermediate nodes.

4.3. \mathcal{N} -hypercube design

Definition 8. The \mathcal{N} -hypercube is a hypercube that consists of **terminal nodes, intermediate nodes and root**. These nodes are connected accordingly to the following rules:

Rule 1: The terminal node can be connected with one intermediate node only.

Rule 2: The root can be connected with two intermediate nodes located symmetrically in opposite faces.

Rule 3: Configurations of the terminal and intermediate nodes on the opposite faces are symmetric, these two faces are connected via the root.

Any n -dimensional structure can be represented by a 3D \mathcal{N} -hypercube which is a hierarchy of $\lceil n/3 \rceil$ of a 3D \mathcal{N} -hypercubes, so that each corresponds to a hypercube representation of a 3-variable Boolean function.

Definition 9. The \mathcal{N} -embedding is a complete binary tree with 2^k terminal nodes embedded into k -dimensional \mathcal{N} -hypercube.

For instance, the 3D \mathcal{N} -hypercube carries information about three-variable Boolean function.

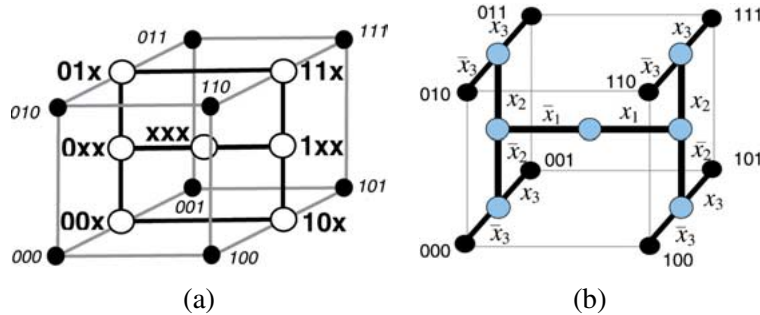


Figure 8. \mathcal{N} -hypercube: coordinate description (a) and embedding of a complete binary tree given in Figure 4 (b).

4.4. Coordinate description

To design and manipulate \mathcal{N} -hypercubes, we need to define a coordinate description of the topological components of \mathcal{N} -hypercubes. For this, we utilize principles of Specification-2 and denote links by $\{0, 1, \mathbf{x}\}$.

There are two possible configurations of the intermediate nodes. The first configuration in coordinate description is:

$$x00 \iff x01 \iff xx1 \iff x11 \iff x10 \iff xx0 \iff x00.$$

The second configuration is relevant to the symmetric face (Figure 8a):

$$00x \iff 0xx \iff 01x \iff 11x \iff 1xx \iff 10x \iff 00x.$$

\mathcal{N} -hypercube includes two configurations with respect to the root:

$$\begin{aligned} \text{Link 1: } & xx0 \iff xxx \iff xx1, \\ \text{Link 2: } & 0xx \iff xxx \iff 1xx. \end{aligned}$$

The coordinate of the root node is xxx . There are two types of links in \mathcal{N} -hypercube: links between terminal nodes and intermediate nodes, and links between intermediate nodes, including the root node.

Example 4. Link $\langle 00,00x \rangle$ indicates the connection of the terminal node 000 and intermediate node $00x$. By analogy, if two intermediate nodes $x10$ and $xx0$ are connected, this is indicated by $\langle x10,xx0 \rangle$ (Figure 8a).

4.5. Relationship between hypercube and \mathcal{N} -hypercube

There is a simple relationship between the classical hypercube utilized in Boolean function manipulation, and \mathcal{N} -hypercube: the coordinates of both hypercubes correspond to each other as follows:

$$\begin{aligned} \text{Link in the hypercube} &\equiv \text{Intermediate node of the } \mathcal{N}\text{-hypercube} \\ \text{Face of the hypercube} &\equiv \text{Face node of the } \mathcal{N}\text{-hypercube} \end{aligned}$$

Example 5. The edges $x00$, $0x0$, $x10$, and $10x$ in Figure 3c correspond to the intermediate nodes $x00$, $0x0$, $x10$, and $10x$ in the \mathcal{N} -hypercube in Figure 8a. Further, the faces $xx0$, $xx1$, $0xx$, $1xx$, $x1x$, and $x0x$ in Figure 3c correspond to the face nodes $0xx$, $1xx$, $x1x$, and $x0x$ in Figure 8a.

4.6. 4D \mathcal{N} -hypercubes

Consider two 3D \mathcal{N} -hypercubes. To design 4D \mathcal{N} -hypercube, those two 3D \mathcal{N} -hypercubes must be linked. Then, the number of bits in the coordinate description of both \mathcal{N} -hypercubes must be increased by one bit on the right (Figure 9). Suppose that \mathcal{N} -hypercubes are connected via link $\langle xxx0,xxx1 \rangle$ between the root nodes $xxx0$ and $xxx1$. The resulted topological structure is

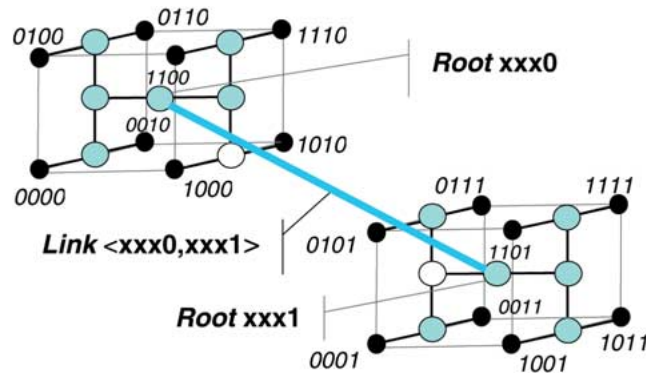


Figure 9. Coordinates of the 4D \mathcal{N} -hypercube.

called 4D \mathcal{N} -hypercube. Thus, the designs of the 4D classical hypercube and \mathcal{N} -hypercube differ.

5. Embedding Binary Decision Tree into \mathcal{N} -hypercube

In this section we will show that an arbitrary binary decision tree, or ROBDD can be embedded into a \mathcal{N} -hypercube. We keep in mind that an arbitrary threshold network can be represented by a binary decision tree or ROBDD, and our final task is to convert these graphical structures into 3D space.

5.1. Basics statements

Briefly, embedding a complete binary tree into a classical hypercube is the mapping of the leaf nodes of the tree with q levels to 2^q nodes of a classical hypercube, where two nodes are adjacent if and only if their q -bit binary codes differ in exactly one position.

Lemma 1. *The number of terminal nodes of the complete binary tree which are mapped into the intermediate nodes of a \mathcal{N} -hypercube is equal to 2^{q-1} , while the number of such embeddings (the number of all wires) is $q \times 2^{q-1}$.*

Proof. The proof follows from the fact that the number of intermediate nodes in the second last level of the binary tree is 2^{q-1} . The number of possible orders of variables is equal to the number of variables, q , so, the number of embeddings is $q \times 2^{q-1}$. \square

Lemma 2. *The number of nodes of the binary tree embedded into the face nodes of a \mathcal{N} -hypercube is equal to 2^{q-2} while the number of such embeddings (the number of all faces) is $q \times 2^{q-2}$.*

The proof is adequate to the proof of Lemma 1.

Corollary 1. *The total number of intermediate nodes and face nodes in a \mathcal{N} -hypercube is $\sum_{i=0}^n 2^{n-i} C_i^n$. The total number of connections between these nodes is $\sum_{i=0}^n 2i \cdot 2^{n-i} C_i^n$.*

It is obvious that the total number of intermediate and face nodes in \mathcal{N} -hypercube is equal to the number of all nodes but leaves in the complete binary decision tree of a Boolean function of n variables, and the total number of connections is equal to the number of edges in the complete binary decision tree.

Theorem 3. *An arbitrary complete 2^q - terminal node binary tree defined by Specification-3 can be embedded into a q -dimensional \mathcal{N} -hypercube.*

Proof. The proof follows from Definition 8, Lemma 1 and 2, and Corollary 1. \square

Theorem 4. *A ROBDD of an arbitrary Boolean function can be embedded into a \mathcal{N} -hypercube.*

Proof. The proof follows from one-to-one correspondence between ROBDD and the complete binary tree. \square

The process of embedding a ROBDD turns to expanding the ROBDD to the complete binary tree, then embedding this tree into an \mathcal{N} -hypercube.

5.2. Algorithm for embedding a binary decision tree into a \mathcal{N} -hypercube

Input data: A complete binary decision tree (Specification-3).

Step 1. Embed 2^q leaves of the binary tree (nodes of level $q - 1$) into a 2^q -node \mathcal{N} -hypercube; assign codes (q -tuple) to the nodes so that each node is adjacent with another one which code differs in exactly one position.

Step 2. Embed 2^{q-1} nodes of the binary tree (nodes of the level $q - 2$) into edges that connect the existing nodes of the \mathcal{N} -hypercube, taking into account the polarity of variables.

Step 3. Embed 2^{q-2} nodes of the binary tree (nodes of the level $q - 3$) into faces of the \mathcal{N} -hypercube, taking into account the polarity of variables.

This recurrent strategy can be used to embed any binary decision tree into a \mathcal{N} -hypercube. Consider $q = 1, 2$ and 3 .

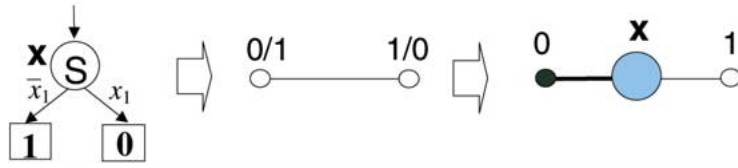


Figure 10. Embedding the 1-variable binary tree into the \mathcal{N} -hypercube

Case $q = 1$. Embedding the binary decision tree that represents a Boolean function of one variable ($q = 1$) is illustrated in Figure 10. The function takes value 1 while $x = 0$ and value 0 while $x = 1$. These values are assigned with two leaves of the tree; it is mapped to the \mathcal{N} -hypercube, which nodes are assigned with the values of variable/function (0/1 and 1/0). Finally, it is mapped to a two-node graph (1-dimensional \mathcal{N} -hypercube) with the node embedded into the edge.

Case $q = 2$. Let us consider a complete binary decision tree that represents a Boolean function of two variables ($q = 2$). It follows from Lemmas 1 and 2 that four leaf nodes of the tree can be embedded into the \mathcal{N} -hypercube with four nodes (a quadrant). The nodes of the tree can be embedded into two opposite edges at a time. The choice of the pair of edges depends solely on the order of variables in the tree. The obtained intermediate nodes are associated with two middle-level nodes of the tree. The face nodes are embedded into the faces, each is connected to both intermediate nodes and is associated with the other variable. Figure 11 illustrates this case. Note, the axes are associated with the polarity of variables (complemented, uncomplemented).

Case $q = 3$. Figure 8b illustrates embedding a binary decision tree of a three-variable Boolean function into a \mathcal{N} -hypercube. First of all, embed 8 leaves of the tree (Gray code ordered). Then, embed four intermediate nodes of the tree into the edges connecting 8 nodes of the \mathcal{N} -hypercube; this is a sub-cube to implement a sub-function with respect to x_3 . Next step, embed two face nodes of the tree into faces of the \mathcal{N} -hypercube and connect them to the intermediate nodes; x_2 is introduced. Finally, embed the root of the tree into the center of the \mathcal{N} -hypercube and connect it to two face nodes; the function is completed with x_1 .

5.3. Connections in n dimensional space

A wire is associated with the tunnel junction controlled by the voltage that is direct (value of x_i) or inverse (value of \bar{x}_i).

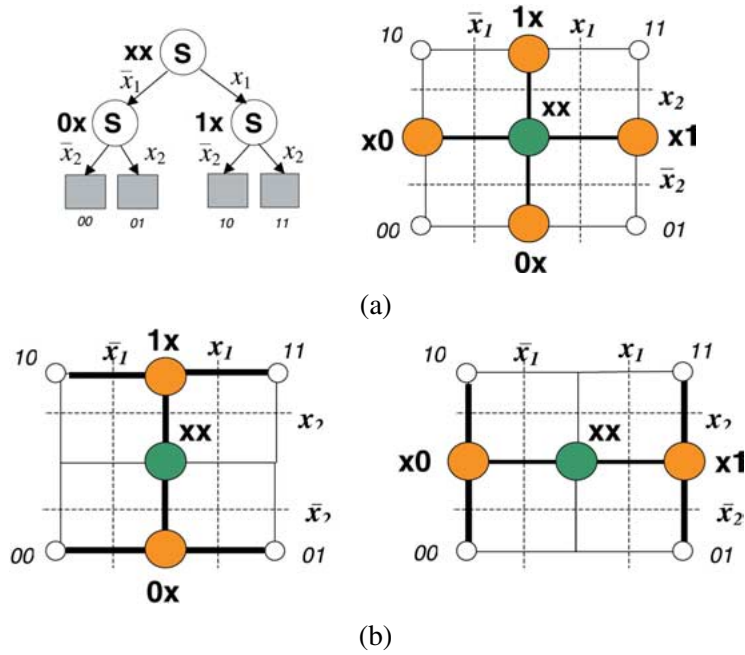


Figure 11. Embedding a two-variable binary tree into a \mathcal{N} -hypercube: (a) two-variable binary tree and the corresponding \mathcal{N} -hypercube (b) possible variants of embedding.

Definition 10. Expanding a \mathcal{N} -hypercubes to more dimensionals is accomplished by:

Rule 1: Connecting roots, and adding one bit on the right of the code for each \mathcal{N} -hypercube.

Rule 2: Assigning coordinates to the new root.

6. Implementation of a Neuron on a Multigate Single-electron Transistor

In this section we show that \mathcal{N} -hypercube is a reasonable model for 3D NN computing. For this we choose a promising nanotechnology, SET technology. The fundamental physical principle of SET is the Coulomb blockade resulting from the quantization of the elementary charge in an isolated node of a double junction structure (Likharev 1999). In a multigate SET, current I_D is a function of input voltages and capacitances:

$$I_D(V_1, V_2, \dots, V_N) = f\left(\sum_{i=1}^N \frac{C_i V_i}{e}\right), \quad (2)$$

where e is a single electron charge ($e = 1.6 \cdot 10^{-19}$ Coulomb). The function (2) indicates that the device implements a neuron function. However, the threshold understanding is different from the traditional one. In a traditional understanding, the neuron output is 1 (a device is on) if the output superceeds some threshold value. In multigate SET, the current I_D takes a minimum when $\sum_{i=1}^N \frac{C_i V_i}{e}$ is an integer because the Coulomb blockade sets in. Here, $\frac{C_i V_i}{e}$ corresponds to the number of exceeds electrons on the i -th input (gate electrode).

When $\sum_{i=1}^N \frac{C_i V_i}{e}$ is a half-integer, the current flows because the Coulomb blockade is lifted. So, the multigate SET will implement the following threshold function

$$I_D(V_1, V_2, \dots, V_N) = f\left(\sum_{i=1}^N \frac{C_i V_i}{e}\right) = \begin{cases} 1, & \sum_{i=1}^N \frac{C_i V_i}{e} = \frac{2l-1}{2}, \\ 0, & \text{otherwise,} \end{cases}$$

where l is an integer, C_i is a weight, and V_i is an input signal.

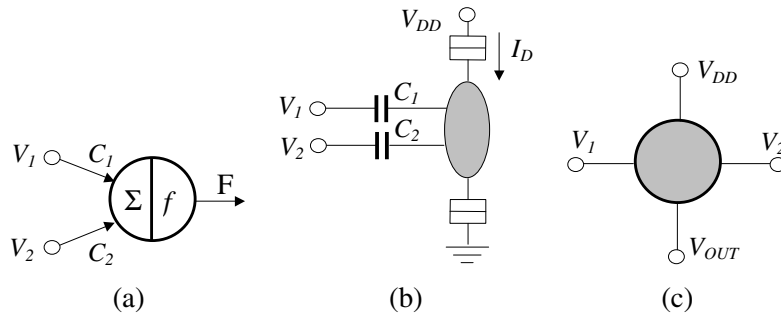


Figure 12. 2-input neuron (a), 2-input multigate SET (b) and 2D two-input neuron model (c)

Consider the 2-input threshold gate (Figure 12a) described by the function

$$F = f\left(\frac{V_1 C_1}{e} + \frac{V_2 C_2}{e}\right).$$

The corresponding multigate SET and its 2D model is given in Figure 12b and Figure 12c accordingly.

The four-input threshold element can be designed based on three multigate SETs. This set of multigate SETs is represented by a complete binary decision diagram that represent a Boolean function of two variables. Embedding this diagram into \mathcal{N} -hypercube results in a 2D \mathcal{N} -hypercube.

7. \mathcal{N} -hypercube Threshold Network Design

To design an arbitrary threshold networks, we have developed a library of a 3D models. This library includes threshold primitives to model logic primitives (AND, OR, NOR, NAND, EXOR), and macro threshold components (full adder, comparator, multiplexer, etc.).

The design strategy to represent a given 2D feedforward NNs in 3D space includes the steps as follows:

- (i) The initial 2D network is leveled.
- (ii) The threshold gates in each level are replaced with the corresponding \mathcal{N} -hypercubes
- (iii) Levels are connected accordingly to the initial netlist

Note that another strategy can be used to partition the initial 2D network, for example, cascading strategy.

8. Experiments

The goal of the experimental study is to represent large NNs in 3D space and evaluate their characteristics such as *space size* and *topological parameters* of nanoICs. Our experiment was organized as follows:

- (i) A common model of NN that is the feedforward (with no feedback) multilayer network with a threshold gate as basic processing unit.
- (ii) The linear threshold gates in this network are described by 3D models (\mathcal{N} -hypercubes) and, thus, by a set of connected \mathcal{N} -hypercubes.
- (iii) A multilevel combinational circuits over the library of NAND, NOR and EXOR gates from the ISCAS85 benchmarks set (combinational circuits)¹ were used.

Notice that we do not consider the problem of optimization of the 3D model.

Experiments were conducted on PIII 800 MHz, 256Mb RAM computer. A fragment of the results is shown in Table 1. The first three columns include the benchmark **Name**, the number of inputs and outputs **I/O**, and the number of gates **#G**. In four columns **Space size** we give the size of nanoICs in **X**, **Y** and **Z** coordinates. Coordinate **X** indicates the number of levels in the network. The value of the resulting nano-solid is calculated as $V = X \times Y \times Z$. The topological characteristics are evaluated by the total number of terminal nodes $\#N_T$, the total number of intermediate nodes $\#N_I$, and the total number of connections $\#C$. The last column contains the runtime in CPU seconds per cube.

Consider, for example, 3D feedforward NN that was designed based on 2D benchmark *c6288*. This 16 bit multiplier (32 inputs and 32 outputs) includes

2416 gates. These gates were grouped in 124 levels that were interpreted as layers. Each gate in i th level c6288 was represented by the corresponding \mathcal{N} -hypercube. Then, the number of incompletely specified \mathcal{N} -hypercubes was minimized. The \mathcal{N} -hypercube in the i -th layer was connected to the corresponding \mathcal{N} -hypercubes in $(i - 1)$ -th and $(i + 1)$ -th layers. The obtained structure was distributed in 3D space as $X \times Y \times Z = 124 \times 2 \times 2$. The value of the resulting nano-solid is equal to $V = X \times Y \times Z = 124 \times 2 \times 2 = 496$ hypercube³. The total number of terminal nodes and intermediate nodes are equal to $\#N_T = 9600$ and $\#N_I = 7184$ correspondingly. To combine all NN layers, 14368 connections were generated. The design of the structure was implemented in $t = 0.93$ seconds.

We observe that the nanodimensional size and the number of nodes are both near linearly proportional to the number of logic gates in the conventional netlist. Notice, that in our experiments we used various metrics to evaluate the designed 3D structure, including the mentioned above traditional parameters (diameter, dilation cost, expansion, load, *etc.*) and specific parameters, in particular, the average number of variables in the logic function described by the \mathcal{N} -hypercube, the average number of links, the average fan-out of the intermediate nodes, and statistical distribution of different parameters.

Table 1. Experimental results on 3D feedforward NN

Name	Test		Space size				Nodes, connections			
	I/O	#G	X	Y	Z	V	$\#N_T$	$\#N_I$	#C	t
c17	5/2	6	3	2	2	12	24	18	36	<0.01
c432	36/7	160	44	8	8	2816	2508	2348	4696	0.06
c499	41/32	202	14	4	4	224	1072	870	1740	0.05
c880	60/26	383	34	2	4	272	1614	1231	2462	0.06
c1355	41/32	546	27	4	4	432	2384	1838	3676	0.17
c1908	33/25	880	51	8	8	3264	5374	4494	8988	0.22
c2670	233/140	1193	50	4	4	800	4842	3649	7298	0.27
c3540	50/22	1669	81	8	8	5184	10258	8589	17178	0.44
c5315	178/123	2307	81	8	8	5184	11728	9421	18842	0.72
c6288	32/32	2416	124	2	2	496	9600	7184	14368	0.93
c7552	207/108	3512	61	4	4	976	14636	11124	22248	1.10

9. Conclusion

The presented design methodology for the design of large-scale feedforward 3D threshold network based on \mathcal{N} -hypercube topology implies 3D representation and spatial input/output interpretation. This is coherent with the requirements, nanotechnology poses to implementation of networks of gates. Taking this into consideration, we believe that SET devices are potential candidates for prototyping the NN in nanodimension. We have demonstrated two strategies for 3D structure design. The first strategy is based on embedding a decision tree or decision diagram into a \mathcal{N} -hypercube:

Decision tree (diagram) \Rightarrow \mathcal{N} -hypercube structure

The second strategy comprises the design procedure over the library of 3D primitives.

Library of 3D primitives \Rightarrow \mathcal{N} -hypercube structure

The primary features of our methodology are the following:

- (i) It offers technology independent data structure compatible with decision diagrams, an advanced structure to represent Boolean functions, and
- (ii) It becomes technology dependent since the decision diagram structures can be directly mapped to the topology of a network of SETs or quantum-dot arrays.

We tested our approach on the large size networks. The reasonable results on space size evaluation, the number of nodes and interconnections have been obtained.

It should be noted, that the proposed design methodology has a generic character. It is possible to generate 3D structures based on another type of decision tree and decision diagram (Bryant and Meinel 2002). In particular, we conducted experiments on so called *linear word-level* decision trees and diagrams (Yanushkevich et al. 2002; Yanushkevich et al. 2004). This data structure provides excellent embedding conditions. However, the resulting 3D structures are technology independent. This is because manipulation of word-level (integer) values is not available in today's nanotechnological solutions.

Acknowledgements

This work was partially supported by NATO through the Collaborative Linkage Grant PST-CLG.979071, and Natural Sciences and Engineering Research Council of Canada (NSERC) through grant 239025-02. The help of

Dr. *P. Dziurzanski* in conducting the experiments is acknowledged. Discussions with Professor Ottar Johnsen, Ecole d'ingénieurs et d'architectes de Fribourg, Switzerland, were extremely useful.

Note

¹ http://www.cbl.ncsu.edu/CBL_Docs/iscas85.html

References

- Asahi, N., Akazawa, M. & Amemiya, Y. (1997). Single-Electron Logic Device Based on the Binary Decision Diagram. *IEEE Transactions on Electron Devices* **44**(7): 1109–1116.
- Bryant, R. E. & Meinel, C. (2002). Ordered Binary Decision Diagrams: Foundations, Applications and Innovations. In Hassoun, S. & Sasao, T. (eds.), *Logic Synthesis and Verification*, 285–307. Kluwer Academic Publishers.
- Fiesler, E. & Beale, R. (eds.) (1997). *Handbook on Neural Computation*. IOP Publishing Ltd and Oxford University Press.
- Frank, M. P. & Knight, T. F., Jr. (1998). Ultimate Theoretical Models of Nanocomputers. *Nanotechnology* (9): 162–176.
- Likharev, K. (1999). Single – Electron Devices and Their Applications. *Proceedings of IEEE* **87**(4): 606–632.
- Okabe, A., Boots, B. & Sugihara, K. (1992). *Spatial Tessellations. Concept and Applications of Voronoi Diagrams*. John Wiley & Sons.
- Roth, J. P. (1999). *Mathematical Design: Building Reliable Complex Computer Systems*. IEEE Press.
- Siu, K-Y., Roychowdhury, V. P. & Kailath, T. (1991). Depth-Size Tradeoffs for Neural Computation. *IEEE Transactions on Computers* **C-40**(12): 1402–1411.
- Saad, Y. & Schultz, M. H. (1988). Topological Properties of Hypercubes. *IEEE Transactions on Computers*. **37**(7): 867–872.
- Takahashi, Y., Fujiwara, A., Ono, Y. & Murase, K. (2000). Silicon Single-Electron Devices and Their Applications. In *Proceedings of 30th IEEE International Symposium on Multiple-Valued Logic*, 411–420.
- Öhring, S. & Das, S. K. (1999). Incomplete Hypercubes: Embeddings of Tree-Related Networks. *Journal of Parallel and Distributed Computing* (26): 36–47.
- Yamada, T., Kinoshita, Y., Kasai, S., Hasegawa, H. & Amemiya, Y. (2001). Quantum – Dot Logic Circuits Based on Shared Binary – Decision Diagram. *Journal Applied Phys., Japan* **40**(7), Part 1: 4485–4488.
- Yanushkevich, S., Shmerko, V., Malyugin, V. & Dziurzanski, P. (2002). Lilarity of Word-Level Models: New Understanding. In *Proceedings of IEEE/ACM 11th International Workshop on Logic and Synthesis*, 67–72. New Orleans.
- Yanushkevich, S., Shmerko, V. & Dziurzanski, P. (2004). Logic Circuit Representation by Linear Decision Diagrams with Extention to Nanostructures. *Automation and Remote Control*. Special Issue on *Arithmetical Logic in Control Systems*. Plenum Academic Publishers, in press.
- Wasshuber, C. (2001). *Computational Single-Electronics*. Springer Verlag.
- Webster, J. (ed.) (1999). *Encyclopedia of Electrical and Electronics Engineering*. John Willey & Sons.