

## Vector-logic computing for faults-as-address deductive simulation

Wajeb Gharibi<sup>1</sup>, Vladimir Hahanov<sup>2</sup>, Svetlana Chumachenko<sup>2</sup>, Eugenia Litvinova<sup>2</sup>, Ivan Hahanov<sup>2</sup>, Irina Hahanova<sup>2</sup>

<sup>1</sup>Computer Science and Electrical Engineering, University of Missouri-Kansas City, MO, USA

<sup>2</sup>Design Automation Department, Kharkiv National University of Radioelectronics, Kharkiv, Ukraine

### Article Info

#### Article history:

Received Dec 3, 2022

Revised Mar 15, 2023

Accepted Mar 27, 2023

#### Keywords:

In-memory computing

Logical vector

Matrix of deductive vectors

Read-write transaction

Sequencer of vector deductive fault simulation

Vector model of input faults

### ABSTRACT

The aim of the research is to create logic-free vector computing, leveraging read-write transactions in memory, to solve the problems of modeling and simulation stuck-at-fault combinations for complex logic elements and digital structures. At the same time, the problem of creating smart data structures based on logical vectors, truth tables, and deductive matrices is considered to simplify algorithms for parallel stuck-at-fault simulation. Vector computing is a computational process based on read-write transactions on bits of a binary vector of functionality, where the input data and faults are the addresses of the bits. A method for the synthesis of deductive vectors for propagating input fault lists is proposed, which has a quadratic computational complexity of read-write transactions. Deductive vectors, combined into a quadratic matrix, represent explicit data structures for parallel simulation of single and multiple stuck-at-faults. The initial information for constructing a deductive matrix is a logical vector and a bit-recoding matrix. Matrix is easily obtained using a recursive procedure based on the combinatorial properties of the truth table. Considering emerging trends, focused on in-memory computing, an algorithm for fault, as addresses, simulation is proposed, using logical and deductive vectors placed in memory. The simulation algorithm is proposed not to use commands of powerful processors.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



### Corresponding Author:

Vladimir Hahanov

Design Automation Department, Kharkiv National University of Radioelectronics

Kharkiv, Ukraine

Email: hahanov@icloud.com

## 1. INTRODUCTION

The motivation of the research is defined by leveraging a vector form of logic [1] organization in-memory computing (simulation). The relevance of this direction can be seen from the latest research by Gartner, which published computational storage (CS) as a trigger trend, which transfers data processing from the central processor to the memory where they are located. Big data should be processed at the place of their storage. The relevance of the application of in-memory computing (IMC), near-memory computing (NMC), and processing-in-memory (PIM) computing is confirmed by [2]–[9]. Creation of a model of deep neural networks (DNN), especially in the field of processing natural language processing (NLP) reduces power consumption by 36.3% and improves big data algorithm performance by 22.6%. The main goal of IMC architectures is to reduce power consumption as much as possible to increase autonomy by reducing data transfer between the memory and the computing unit using bitwise logical operations (NOT, AND, OR, and XOR) inside and next to the memory array, promoting the concept of in-memory computing (CiM). Bandwidth and power consumption have become the most critical bottleneck in von Neumann's computing architecture

due to the separation of processor and memory. The realization of computing unity and memory in one place opened a promising direction for research in-memory computing (CIM). Techniques that bring computation as close as possible to the memory array, such as in-memory computation (IMC), near-memory computation (NMC), and in-memory processing (PIM), can reduce the cost of moving data between the core and memory. In-Memory Computing delivers speed and up to 78% energy savings. This type of IMC-computing is devoted to more than 64,000 publications in IEEE Xplore, which indicates the relevance of the problem of big data analysis using simple CPU-free models placed in memory. In-data (in memory) computing is a computational process based on read-write (logic) transactions in memory, minimizing time and energy consumption for processing big data.

The idea of the study is based on the superposition of components: in memory computing, the use of vector data to describe the logical functionality and read-write transactions instead of a powerful processor to simulate single and multiple faults, like addresses. This makes it possible to increase the speed of deductive simulation, and transfer of computing to a lower level of computational processes (read-write transaction), where the von Neumann architecture and the post-Jablonski theorem about functional completeness can be ignored [10]–[12]. Vector computing is based on read-write transactions on vector data structures in address memory. The relevance of this direction can be seen from the latest Gartner Hype Cycle, which highlighted computational storage (CS) as a trigger trend of transferring data processing from the CPU to the memory where they are located [13]. Big data must be processed where it is stored.

Vector computing is a computational process based on read-write transactions on the bits of a binary vector of functionality that forms computational storage, where the input data (conventional memory) are bit addresses. Data (fault vectors) in the proposed vector-deductive simulation method are used as addresses for processing the data itself.

The input data models can be represented in several ways, as shown in Figure 1: i) sets, which are compact data that require a complex and sequential algorithm for processing inputs, ii) vectors, which provide unitary data coding, use a parallel register algorithm for data processing and sequential algorithm for processing inputs, iii) addresses, which provide a compact encoding of unitary data and a sequential algorithm for their processing by read-write transactions in memory free of logic and processor with parallelism in address columns. Based on the associative law, it is possible to increase the number of input variables by expanding the vector memory elements:  $((x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8)x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15})x_{16}x_{17}x_{18}x_{19}x_{20}x_{21}x_{22}$ . The vector  $Q = 01101001$  is a function of three variables that form addresses from the data to be analyzed.

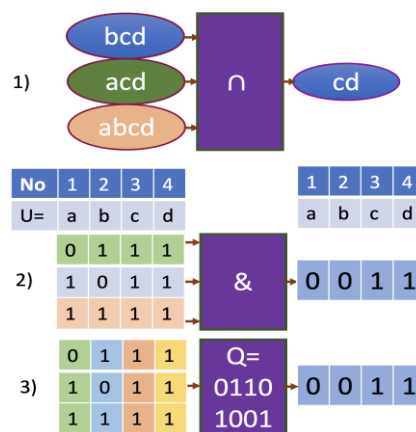


Figure 1. Input data models

In design and testing, all problems are solved based on project representation using standard hardware description languages (VHDL, Verilog, System C). They allow you to combine the efforts of many developers and link individual projects into a system. It is possible to use powerful compilers for the synthesis and analysis of digital projects. All industrial systems have data structures that are the strictest secret of all leading companies in the EDA market. This is because the data structures determine the efficiency of compilers and synthesizers by 80%, which cost a lot of money in the EDA market. If we talk about the classification of software products created for the EDA market, these are, first, simulation of correct behavior, fault modeling, synthesis of digital projects and their verification by logical and temporal parameters, diagnosing defects, and building tests. There are hundreds of companies in the EDA market dealing with these issues. The largest companies are Cadence and Synopsys.

In design and testing, three main forms of describing models are used: tabular, analytical, graph [10]–[21]. Analytical and graph requires for its interpretation complex solvers and a powerful system of processor instructions. Tabular models are an explicit form of specifying functionality and therefore do not require processor instructions for their interpretation. At the same time, a table and a vector are two forms of description of functionalities focused on memory computing that pass into each other. The logical binary vector is a compact view of the truth table in the form, ordered by the addresses of the sequence of bits, as the output states of the functionality [11]–[20]. Here and below, the logical vector passes from a part of the truth table into an independent compact form to set the functionality to create a vector logical in - memory computing for processing big data.

The main problems of fault simulation are: i) The combinatorial analysis algorithms are characterized by high computational complexity at the register level of model description. ii) The algorithms for modeling and simulating sequential circuits are associated with an unpredictable number of iterations [11]. iii) A significant amount of data structures for the analysis of digital systems-on-chips negatively affects the performance of fault simulation methods and test synthesis [15]–[22]. iv) Fault simulation algorithms for high-dimensional logic or the analysis of circuits with converging branches are complex [10]. v) The parallel solution of the problems of modeling and simulation of functionalities and digital structures is problematic [9]–[12]. vi) In addition, the crisis of modern computing is associated with two problems: a long time for analyzing big data on a processor-memory pair, as well as a catastrophic increase in power consumption for processing big data on modern microelectronics of powerful processors. Further, an original solution to the indicated problem is proposed based on the axioms of in-memory vector-logical computing: i) All data is in memory. There is no functionality or structure that cannot be implemented as logical vectors in memory. ii) There is no data that cannot be used as addresses for processing by vector logic in memory. iii) There is no computational process that could not be implemented using read-write transactions in memory. iv) The most technologically advanced and smart structure for read-write transactions to analyze big data is a vector. v) The logical vector or truth table contains addresses as explicit solutions to any combinatorial problem, including big data analysis.

The goal is to develop a vector-deductive method for modeling faults, as addresses, for the analysis of logical functionalities and circuits of any dimension. Tasks are i) the development of a vector method for the synthesis of deductive matrices for transporting input faults to the output of an element, ii) the use of a logical vector for parallel simulation of faults in digital functionality, and iii) the use of a logical vector for deductive modeling and simulation of faults in a digital circuit.

The problem of transferring the von Neumann architecture to memory and replacing a powerful read-write processor with transactions on logical vectors is solved to reduce energy and time costs when modeling and simulation of logical functionality of any dimension. The object of research is in-memory computing, which reduces energy and time costs when processing big data. The subject of research is in-memory modeling of SoC logical components, of any dimension, using read-write transactions on logical vectors. The main idea of the fault modeling method is to develop sequencer blocks for vector-deductive modeling of single and multiple constant faults. The research formula is an in-memory fault as address simulation via read-write transactions on logic vectors. The metric of a vector-deductive simple and reliable simulator for processing vectors of single stuck-at faults can be represented by Figure 2.

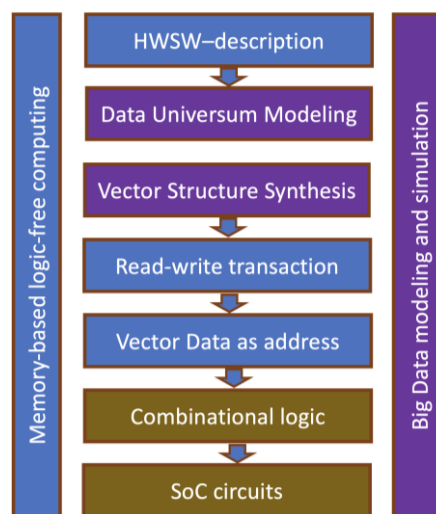


Figure 2. Metric of the vector-deductive simulator

The problem of the von Neumann machine is solved, which consists of the exchange of data between the processor and memory, which causes a significant increase in data processing delay and energy consumption. It is proposed to replace the powerful command system of the central processor with read-write in-memory transactions for processing data like addresses. Smaller computing element is more efficient for big data analysis. The essence of the solution to the problem is the creation of in-memory read-write computing to simulate faults for verification of the quality of IP-core SoC tests. In-memory fault simulation technologies are designed to reduce computing latency and improve energy efficiency.

## 2. DEDUCTIVE VECTOR MATRIX SYNTHESIS METHOD FOR FAULT SIMULATION

The equation  $T \oplus F \oplus L = 0$  conceptually solves all problems of modern deterministic and probabilistic computing. The equations allow for solving three classical problems. The first one is  $T = F \oplus L$ , which is the test generation based on the model for a given fault list. The second one is  $F = T \oplus L$ , which is the synthesis of a reference model of functionality through faults and tests of the device. The third problem is  $L = F \oplus T$ , which is detecting and diagnosing faults of a device on a given test. Including the computing equation solves the problem of deductive modeling and simulation of a digital project when models of digital devices are presented in any of three types: analytical, tabular, and vector. Further, only the vector form of representation of functionalities is used, which differs from Armstrong's method [23], which used the analytical form of representation of digital structures. The articles [19]–[21], [24]–[29] are free from analytical forms of specifying logical functionality that require complex solvers of logical equations, as well as the presence of a processor with a powerful command system for implementing algorithms. The novelty of the proposed research is to replace the analytical models for fault simulation with vector ones [10], [14]–[18], which must be implemented in memory. Further, it is proposed to use a logical vector for the synthesis of a deductive matrix, which is used to transport the input lists of faults to the output of a logical element, of any complexity. In this case, the synthesis of deductive analytical forms is not used, it is separated from simple procedures for analyzing smart data structures, which include logical vectors, deductive matrices, and truth tables for recording combinations of faults in them [10], [14]–[22], [24]. At the same time, the procedures for analyzing deductive matrices are parallel, where the degree of parallelism depends on the complexity of the functional element. The more variables a logic element has, the greater the degree of parallelism it has. In this case, the truth table is an ideal model for storing and modeling and simulation of the combination of single and multiple faults.

The method for synthesizing deductive vectors using a Q-vector has two steps. The first step is the modification of the Q-vector of the element, as shown in Figure 1, on the input i-set according to the rule:  $L = Q \oplus Y_i$ . Here,  $Y_i$  is the state of the logical element on the input set  $x_i$ . The second step is the determination of the deductive vector for the i-input set by  $D_j = L_{H_{ij}}, j = \overline{1, 2^n}$ , which permutes the bits according to the permutation matrix H, which can be easily obtained through recursion.

$$H_{ij}(1,2,3) = \begin{bmatrix} 01 \\ 10 \end{bmatrix} \rightarrow \begin{bmatrix} 01 & 23 \\ 10 & 32 \end{bmatrix} \rightarrow \begin{bmatrix} 01 & 23 & 45 & 67 \\ 10 & 32 & 54 & 76 \\ 23 & 01 & 67 & 45 \\ 32 & 10 & 76 & 54 \\ 45 & 67 & 01 & 23 \\ 54 & 76 & 10 & 32 \\ 67 & 45 & 23 & 01 \\ 76 & 54 & 32 & 10 \end{bmatrix}$$

The algorithm ends when all deductive vectors for all input  $2^n$  sets have been generated. Thus, the matrix of deductive vectors is obtained based on the execution of the operator  $D = (Q \oplus Y)_H$  that is a result of the superposition of the operators:  $L = Q \oplus Y$  and  $D = L_H$ . The synthesis of matrices of deductive vectors for based 2-input logic is shown in Figure 3.

The synthesis of deductive formulas for mutually inverse elements:  $Q = 0110$  and  $Q = 1001$  gives the same values of the matrix of deductive vectors, which degenerate into one vector 0110 on all input sets. The computational complexity of executing this operator is  $C = 2 \times 2^n \times 2^n = 2^{2n+1}$ . In the case of parallel execution of register operations on vectors, the computational complexity of this operator will be equal to  $C = 2 \times 2^n$ .

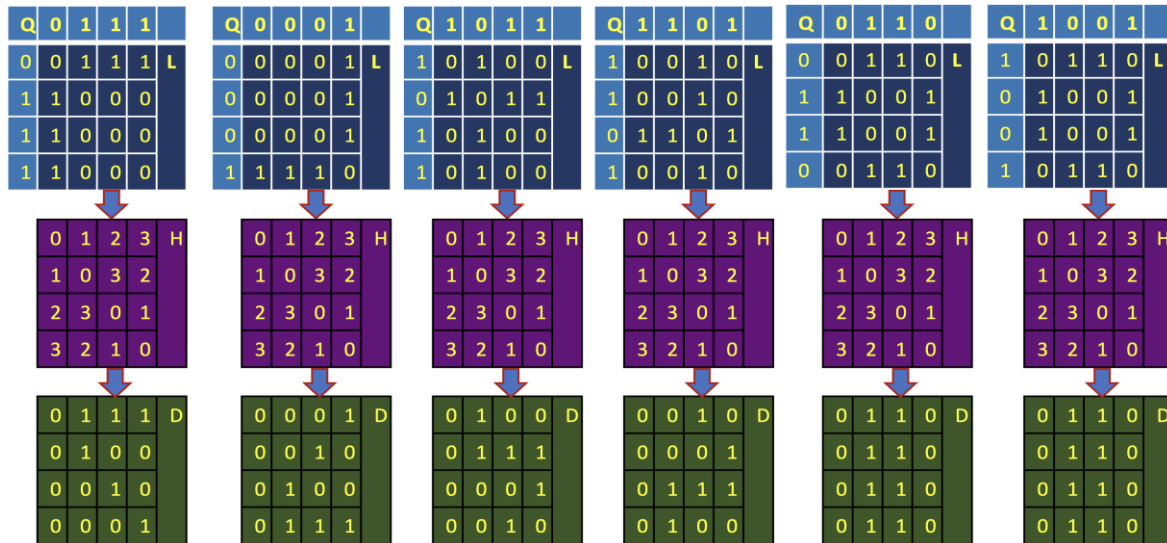


Figure 3. Synthesis of deductive vectors for based 2-input logic

### 3. DEDUCTIVE ANALYSIS OF A LOGIC CIRCUIT

Verification of the deductive fault simulation method was carried out based on the digital structure having four logical elements as shown in Figure 4. For these elements, deductive matrices were created for transporting faults from inputs to output based on logical vectors (1011, 0110, 0111, 1101). Then, the circuit was simulated on the test set  $X=01101$  in order to transport fault lists  $L=\{L1, L2, L3, L4, L5\}$  to the circuit output  $L9 = F[X, L(x_i)]$ .

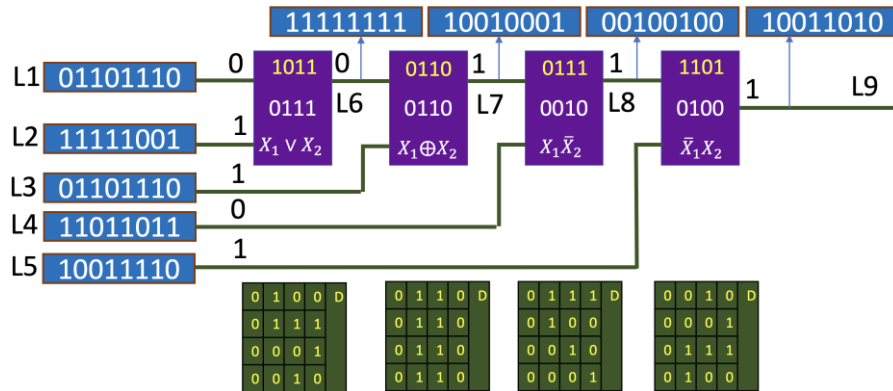


Figure 4. Digital structure of four elements

To solve this problem, it is necessary to use deductive vectors, which are built based on the logical vector of functionalities. Each element of the circuit contains a functionality vector  $Q$ , as well as vector  $D$  (white color), which specifies the procedure for propagating input fault lists to the output for a given input vector. In this case, pairs of input fault-vector coordinates of the same name are considered as addresses for reading a bit of the deductive vector to the output of the element. The read bits sequentially form a vector of output faults propagated from the element inputs. How does the proposed research differ from publications that are devoted to fault modeling methods [10], [14]–[22], [24]? A few major differences are that i) vector tabular deductive fault simulation uses smart data structures based on logical vectors; ii) instead of a powerful processor, read-write transactions in memory are used to implement the algorithm; iii) the method allows parallel processing of single and multiple faults, which are placed in truth tables; and iv) the method is focused on processing IP-core SoC functionality for modern digital devices under the control of the IEEE 1500 SECT standard.

#### 4. DEDUCTIVE MATRIX SYNTHESIS OF GATE AND RTL LOGIC

Also of interest is the process of creating deductive models of the main logic elements, which can be used as a library for creating and analyzing circuits. Below are tables for the synthesis of deductive vectors to check the quality of tests of logical circuits. The most primitive elements are the inverter ( $Q=10$ ) and the repeater ( $Q=01$ ). Even though these are different elements, they have the same deductive vectors, which allow providing digital logical activity to transport the fault vector from input to output without distorting it as in Figure 5.

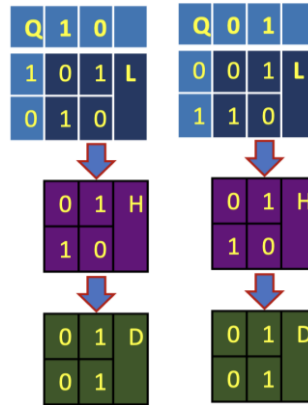


Figure 5. Synthesis of deductive vectors for primary mutually inverse elements: 10 and 01

The next frame is devoted to the process of synthesizing deductive formulas for a three-input logic element given by vector coverage 10000001 as shown in Figure 6. Such an element should be considered as a black box or RTL-level function representation in relation to its structure, which can be implemented differently when specifying its behavior in a vector. Here, the result of propagating lists of activities from the input to the output of this element is of interest. In this case, it is not interesting which paths are involved within a particular implementation of a logical element. Nevertheless, the synthesis of deductive formulas for this element showed that on all input actions, the activity propagation formula has the same value on pairs of sets: 1–16 and 5–6. The remaining sets, having symmetry, are not repeated in the matrix of deductive vectors. Here and below, the zero coordinates of the matrices L and D are represented by empty cells for the purpose of figurative perception of information.

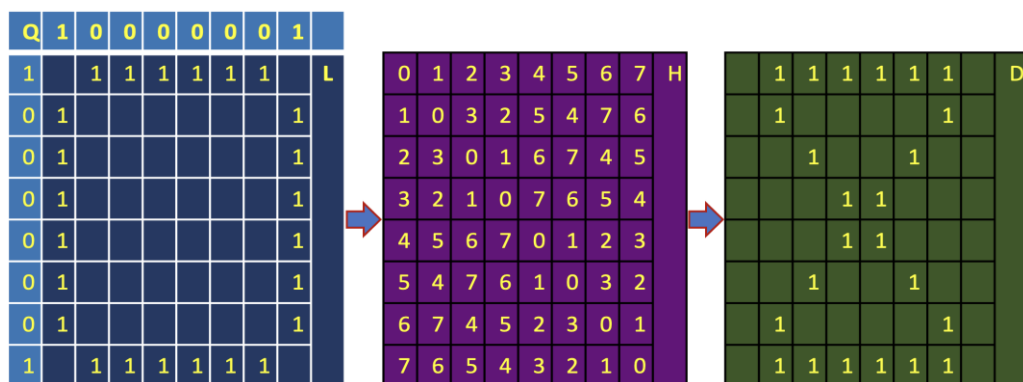


Figure 6. Synthesis of deductive vectors for a three-input 10000001–element

The circuit in Figure 7 has the property that the deductive vectors in the generated MDV matrix are the same and equal to 00110011. The analytic form of such a vector after elementary transformations is  $D=X_2$  on all input sets. This means that three of the four functionality variables are non-essential and cannot be activated by input faults.

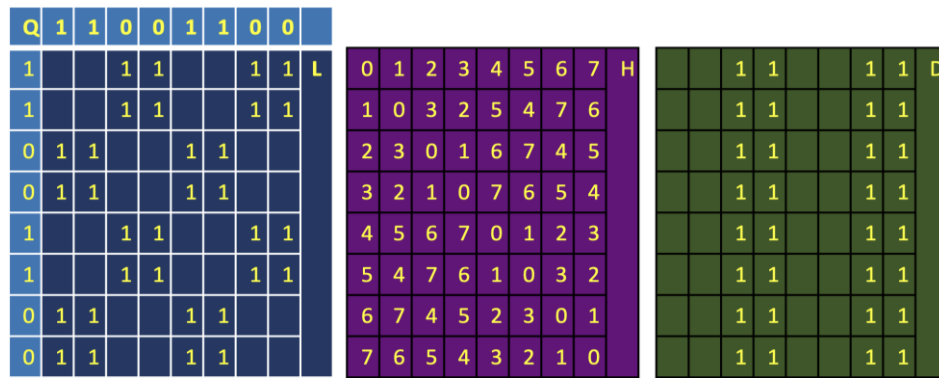


Figure 7. Synthesis of deductive vectors for a three-input 11001100–element.

The next frame is devoted to the process of synthesizing deductive formulas for a 4-input logic circuit, the gate structure of which is known as the “Schneider circuit” [21], which is shown in Figure 8. RTL-model of the circuit is specified by vector coverage  $Q=10000000000000001$ , shown in Figure 9. Naturally, for each deductive vector, we can obtain an analytical form in the form of a DNF for propagating faults on a specific set. But this way is technologically complex and computationally expensive, which means it is not applicable to the market of electronic technologies.

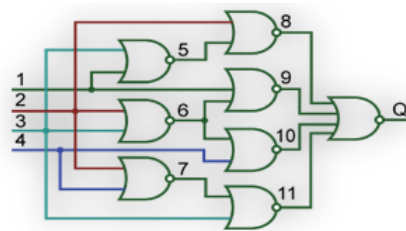


Figure 8. Gate implementation of the 4-input Schneider circuit

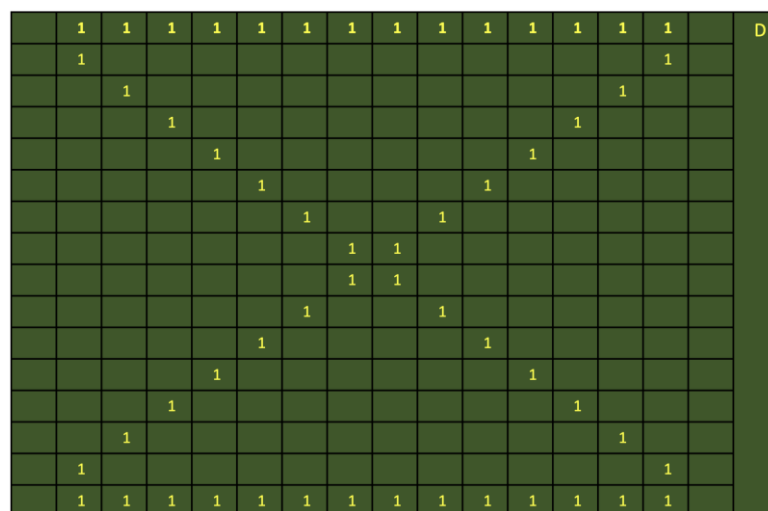
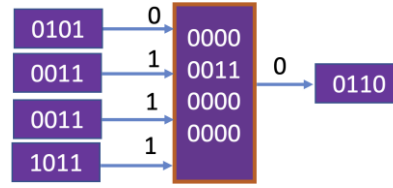


Figure 9. Synthesis of deductive vectors for 4-input circuit Q=1000000000000001

An example of simulating input faults of a 4-input element on the deductive vector 0000001100000000 (line 7 of the above MDV) is shown in Figure 10. For four automatic cycles, the vectors of input faults were propagated to the output with the result obtained in the form of one vector of output faults 0110. Solving this problem on the gate structure of this circuit would require 32 automaton cycles.



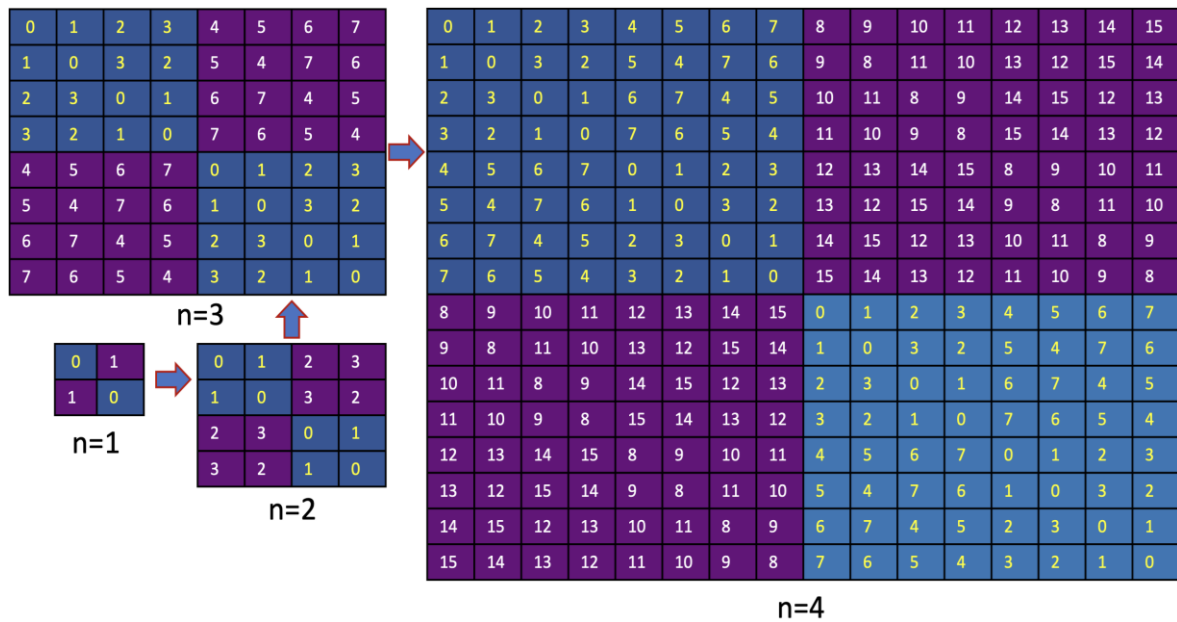
Figure 10. Simulation of 4-input circuit  $Q=1000000000000001$ 

Increasing the number of inputs of RTL functional element leads to an increase in simulation performance since more input fault vectors are simulated in parallel to obtain an output fault vector. Using an 8-input element improves simulation performance by a factor of 11 compared to its structural gate equivalent. In the general case, the improvement in simulation performance for RTL circuits having  $n$  inputs, compared with the analysis of the DNF structure of two-input gates, is given by  $Q = \frac{n}{2} + n - 1$ , with  $n=4, 8, 16 \dots$

Of interest is the technological simplicity of the algorithm for synthesizing the H-matrix of coordinate permutation  $H^i = \begin{bmatrix} H_1^{i-1} & H_2^i \\ H_3^i & H_4^{i-1} \end{bmatrix}$ ,  $i=1,2,3 \dots$  is the number of input variables, which has 3 items.

- The first and fourth quarters of the matrix are taken from the previous recursive calculation of the matrix for  $n=i-1$  variable, here the equalities  $H_1^{i-1} = H_4^{i-1}$ ,  $H_2^{i-1} = H_3^{i-1}$  are satisfied.
- The second quarter of the matrix, which is depicted in Figure 11, is found based on the expression  $H_{i,j+2^{n-1}} = (2^n - 1) - H_{i,2^{n-1}-j}$ ,  $j = 0, 2^{n-1}-1$ ,  $i = 0, 2^{n-1}-1$ .
- The third quarter of the matrix is found by copying the second part of the matrix into the third area  $H_3^i = H_2^i$ .

Despite the technological simplicity of the proposed algorithm for generating a matrix of deductive vectors, this approach has an obvious drawback associated with the dimension of the tables with many input variables. It can be eliminated if only one deductive vector is promptly generated on the input test set to simulate faults. To do this, you just need to use a single operator  $D_i = (Q \oplus Y_i)_{H_{ij}}$ , described earlier. The computational complexity of this procedure is equal to  $2^n$ ,  $n$  is the number of variables in the logical element, which is determined by permuting the bits in the  $Q$ -vector of the H-matrix to obtain the  $D$ -vector. In this case, the deductive simulation of faults will not differ much in speed from the fault-free simulation of a digital circuit. The processing time delta of one logic element  $\Delta T = TD - TG = tD + tF = 2^{n+1}$  will be represented by the deductive vector generation time  $tD$  and plus the processing time of input fault vectors  $tF$ .

Figure 11. Synthesis of vector recoding matrix  $D=L(H)$



## 5. VECTOR-DEDUCTIVE SEQUENCER

The matrix of deductive vectors is a certain redundancy of a digital project, which is the cost for a fast and technological solution to the problem of assessing the quality of test patterns and generating a fault functions table to detect faults at the stage of functioning a digital product. The proposed deductive matrix has six properties, which are compactness, parallel data processing, technological placement in address memory, data uniformity in size and properties, simultaneous simulation of fault-free behavior of the element and all faults of previous elements and focus on the technological solution of the problems of simulation, testing, and diagnostics of any logical systems.

The vector structure of deductive modeling and simulation in-memory, as depicted in Figure 12, is the simplest implementation of the computing device for deductive fault simulation of a digital circuit (element) to assess the quality of the tests in the class of single stuck-at faults. The main and single memory block stores a matrix of deductive vectors, which has a dimension of  $2^n \times 2^n$ , where the first number is the number of deductive vectors for a logical element of  $n$ -variables, and the second one is the dimension of each vector. Therefore, the first macro input of the memory block Vector address has  $n$  binary variables that allow addressing any of the  $2^n$  deductive vectors of the matrix. The second macro input of the memory block byte address also has  $n$ , but already registers binary variables, which allows addressing any of the  $2^n$  bits of the vector, selected by the first macro input, by their binary combination. The output of the memory block fault list has a bit width equal to the second macro input, determined by the power of the simulated input faults that must be propagated through the element of the digital structure.

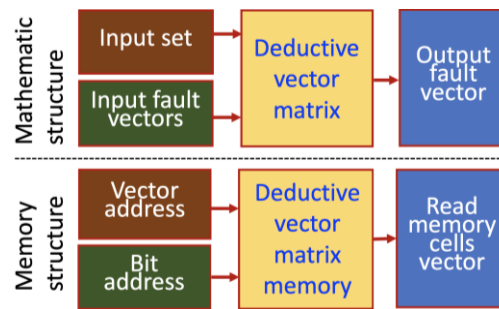


Figure 12. Vector sequencer of deductive simulation implemented in the memory block

There is a one-to-one correspondence between the components of the two schemes (Mathematic structure  $\rightarrow$  Memory structure): Input set  $\rightarrow$  Vector address, Input fault vectors  $\rightarrow$  Bit address, Deductive vector matrix  $\rightarrow$  Deductive vector matrix memory, Output fault vector  $\rightarrow$  Read memory cells vector. Here, the input binary set (data) entered in the gate is interpreted as the address to access the deductive vector in matrix memory. The vector of output faults propagated from element inputs is formed by reading the coordinates of a deductive vector placed in matrix memory. Unusual or paradoxical is the fact that combinations of bits of input fault vectors (data) act as addresses of deductive vector bits for reading them from memory. Simply put, the faults (input data) are used as addresses to read the deductive vector bits from the matrix memory to form the output fault list.

An example of the operation of the deductive fault simulation sequencer on the memory block is shown in Figure 13. The first two logical inputs with their values  $x_1x_2=10$  form the address of the deductive vector 0010. After that, each pair of input binary signals on the vectors  $X_1X_2$  forms the bit address of the previously selected vector. The content of the selected bit at each simulation cycle forms the bit of the output fault vector obtained because of propagating the input fault lists through the element.



Figure 13. An example of deductive simulation on a memory block

For example, the first signal pair  $X_1X_2=00$  addresses the 0-th memory cell of the vector 0010 (in cell number format: 0123), where 0 is located, forming the first cell of the output fault vector. The second pair of

signals  $X_1X_2=10$  addresses the second memory cell of the vector 0010, where 1 is located, forming the second cell of the output fault vector. Thus, pairs of signals in the same-named bits of the input fault vectors play the role of the cell address of the selected deductive vector of the matrix. The number of simulation cycles is equal to the power of the fault vectors of the input variables. In this case, there will be eight simulation cycles, which form eight bits of the output fault vector. Deductive-vector fault simulation of an arbitrarily complex digital circuit is reduced to a primitive memory transaction represented by an addressable read-write operation. No traditional logic is required. The same result was obtained earlier for simulating fault-free behavior of digital systems [13], [25], [27], [29], where the vectors of logic element output states were used as descriptions of them. The structure of the device for vector-deductive fault modeling and simulation, which is proposed in the work, has some differences from analogs: vector-logical assignment of functionalities, the use of read-write transactions to implement the in-memory algorithm, and smart data structures that simplify the fault modeling and simulation procedures.

The metric of a functional  $n$ -input element  $F$ , represented by a vector of  $2^n$  states of output coordinates and its deductive model  $DF$ , represented by an MDV matrix of  $2^n$  deductive vectors with a dimension of  $2^n$  binary coordinates, is shown in Figure 14. Here, the Bit address for the deductive element MDV is formed by the same-named coordinates or bits of the input fault vectors, and the Vector address is formed by the input binary set entered on the inputs of the functional element. The bit address for the functional element  $F$  is formed by the input binary word or set.

Thus, the ratio of the two models by memory is  $F/DF=2^n/2^n \times 2^n=1/2^n$ . The computational complexity of synthesizing a matrix of deductive vectors is estimated by  $Q=n \times 2^n + 2^n \times 2^n$ , the first term determines the complexity of generating address-disordered input variables of the deductive vectors in register operations on vectors, the second term determines the complexity of the coordinate operations for reducing the bits of deductive vectors according to the order of the binary addresses, composed by input variables. The computational complexity of the analysis of the matrix of deductive vectors when performing deductive simulation is  $Q=k \times R$ , where  $k$  is the dimension of input fault vectors,  $R$  is the duration of the operation of reading the contents of the addressable bit of the deductive vector from the MDV memory. The significant deductive redundancy of the project pays for the quality and reliability of the digital system, which also allows for generating tests, evaluating their quality, and solving any problems related to diagnosing faults in the design and operation of a digital device in critical areas of human activity.

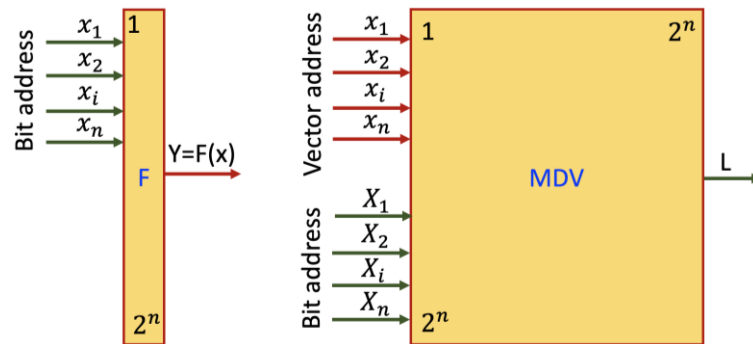


Figure 14. Metrics of functional and deductive elements

## 6. VERIFICATION OF THE VECTOR-DEDUCTIVE METHOD FOR FAULT SIMULATION

The scheme for verifying the simulation method is taken from library C17, as in Figure 15, which has converging fanouts (2,7,8). The input set 11111 was simulated. Each circuit element has a logical vector equal to  $Q=1110$ . Based on this logical vector, a deductive simulation matrix is built, which is determined by four vectors: 0001, 0010, 0100, and 0111. The simulation results are presented in Figure 16. Only single fault constants on the input lines of the elements and the circuit were considered. The lines of the table show the defects that are checked by the test on the corresponding lines of the circuit. The fault that is checked by the test will always be inverse with respect to the healthy state of this line. The last line of the table contains actual defects, which are checked by the input text set 11111. Different variants of deductive fault simulation are presented in [25], [27], [29].

It should be borne in mind that the choice of the deductive vector of each element is based on the input binary word of the input variables  $x_1x_2$ . There are four such words for each element: (00,01,10,11), according to the matching number of deductive vectors. The simulation table has empty coordinates, which

correspond to the 0<sup>th</sup> value of the signals. To explain the procedure for using the table for manually modeling output fault lists, the diagram in Figure 17. Here, fault lists are simulated as column addresses, with preselection of deductive vector number three.

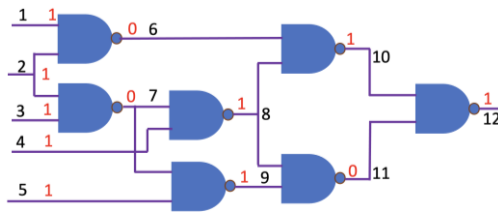


Figure 15. Scheme C17 with reconvergent fanouts

	1	2	3	4	5	6	7	8	9	10	11	12
States	1	1	1	1	1	0	0	1	1	1	0	1
1	1											
2		1										
3			1									
4				1								
5					1							
6	1	1				1						
7		1	1				1	1				
8		1	1				1	1				
9		1	1				1		1			
10	1	1				1				1		
11	1	1	1				1	1	1		1	
12			1				1	1	1		1	1
Faults			0				1	0	0		1	0

Figure 16. Fault simulation table for circuit C17, input set 11111



Figure 17. Simulation of faults as addresses for the formation of fault list for the 12-output circuit

The computational complexity of the proposed deductive vector method is determined by  $Q = \frac{1}{2} \times k \times n^2$ , where  $k$  is the data read from memory,  $n$  is the complexity of the circuit, determined by the total number of input, internal and output lines,  $\frac{1}{2}$  is part of the table modeling data that needs to be processed. Figure 18 shows the simulation of faults as addresses for the input set 11001. The red color along the diagonal of the table marks the ones that always identify the faults checked at the output of each element.

In Figure 19, one procedure shows the results of combining two simulation iterations of two test cases: 11111 and 11001. These results give a total of  $9/24=37.5\%$  of tested faults. In general, a test is considered good if it checks for more than 95% of the faults on the circuit lines. On each line of the circuit, the stuck-at-1 and stuck-at-0 must be detected, which together, when combined, gives the symbol  $x = \{0,1\}$  denotes a detection on the line  $\equiv 0$  and  $\equiv 1$ . Thus, the verification of the fault simulation method as addressed is shown here. The method is easy to implement in memory and the construction of vector logical data structures for fault modeling does not require complex expensive synthesis. Also, the method does not require a powerful command system of a universal processor, but it can be easily managed using read-write transactions in memory. The method is economical in terms of energy consumption and the time of the stuck-at-fault simulation.

	1	2	3	4	5	6	7	8	9	10	11	12
States	1	1	0	0	1	0	1	1	0	1	1	0
1	1											
2		1										
3			1									
4				1								
5					1							
6	1	1				1						
7		0	1				1					
8		0	0	1			0	1				
9		1	1	1			1		1			
10	1	1	0	0		1		0		1		
11	0	1	1	0			1	0	1		1	
12	1	1	1	0		1	1	0	1	1	1	1
Faults	0	0	1			1	0		1	0	0	1

Figure 18. Vector–deductive simulation of a circuit on an input set 11001

Faults	1	2	3	4	5	6	7	8	9	10	11	12
11111			0				1	0	0		1	0
11001	0	0	1			1	0		1	0	0	1
Coverage	0	0	x			1	x	0	x	0	x	x

Figure 19. Fault coverage matrix

The problem of the von Neumann machine is solved, which consists of the exchange of data between the processor and memory, which causes a significant increase in data processing delay and energy consumption. It is proposed to replace the powerful command system of the central processor with read-write in-memory transactions for processing data like addresses. Smaller computing element is more efficient for big data analysis. The essence of the solution to the problem is the creation of in-memory read-write computing to simulate faults for verification of the quality of IP-core SoC tests. In-memory fault simulation technologies are designed to reduce computing latency and improve energy efficiency.

Comparison with existing analogs in the field of fault modeling is carried out according to the time-money-quality metric [10], [15], [17], [18], [30]–[35], [36]–[45], [46]–[51]: i) The complexity of the algorithm implementation. Here, instead of a powerful system of more than 300 processor instructions, one instruction for read-write transactions over memory is used. This is a plus. ii) Redundancy of data structures. The data structures of the explicit vector assignment of logical functions and the matrix of deductive vectors are used. This is a minus. iii) The computational complexity of simulation faults, as addressed, does not depend on the number of gate inputs as in other simulation systems. This is a plus. iv) The computational complexity of algorithms for processing smart data structures for simulating faults as addresses is determined by the components  $Q = \frac{1}{2} \times k \times n^2$ , where  $k$  is the time to read a vector bit from memory;  $n$  is the number of lines in the circuit;  $1/2$  is half of the simulation table. Approximately the same estimates have all existing industrial analogs of fault simulation. So, it can be concluded that the proposed in-memory computing technology focuses on embedded simulation for servicing IP-core SoC.

## 7. CONCLUSION

The scientific novelty of the proposed research in-memory vector fault, as address simulation is formed by some components: First, the computing equation  $T \oplus F \oplus L = 0$  is used to solve the problems of modeling and simulation single and multiple stack-at-fault. Second, built a model of smart data structures for in-memory computing, which include logical vectors of the truth table and deductive matrix. Third, algorithms for parallel analysis of smart data structures have been developed to simulate faults as addresses of the in-memory truth table. Fourth, efficient procedures for constructing a matrix of deductive vectors based on elementary transactions in memory have been developed. Fifth, data structures, methods, and procedures are verified based on fault simulation for complex logic elements and digital circuits. Sixth, the method can be

used for parallel processing big data, which is interpreted as cell addresses of deductive and/or logical vectors that compose the computational memory on which read-write transactions are performed and no logic. The method can be the basis for a new deterministic quantum logic-free computing based on the execution of photonic (quantum) transactions on a structure of stable subatomic particles considered as memory. In addition, the proposed method can effectively solve the problem of recognizing any activity in the cyber-physical space. Seventh, the implementation of vector deductive logic models in the FPGA LUT will allow one to obtain the performance of fault simulation of real SoC digital blocks at the level of hundreds of nanoseconds.




## REFERENCES

- [1] D. Devadze, Z. Davitadze, and A. Hahanova, "Vector-deductive memory-based transactions for fault-as-address simulation," in *2022 12th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Dec. 2022, pp. 1–6. doi: 10.1109/DESSERT58054.2022.10018769.
- [2] S.-H. Chang, C.-N. J. Liu, and A. Kuster, "Behavioral level simulation framework to support error-aware CNN training with in-memory computing," in *2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, Jun. 2022, pp. 1–4. doi: 10.1109/SMACD55068.2022.9816307.
- [3] M. Izhar, S. Ahmad, and S. A. Rahman, "Logic circuit implementation for enabling SRAM based in memory computing," in *2022 5th International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT)*, Nov. 2022, pp. 1–5. doi: 10.1109/IMPACT55510.2022.10029072.
- [4] H. Amrouch, N. Du, A. Gebregiorgis, S. Hamdioui, and I. Polian, "Towards reliable in-memory computing: From emerging devices to post-von-neumann architectures," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2021, pp. 1–6. doi: 10.1109/VLSI-SoC53125.2021.9606966.
- [5] K.-H. Li, C.-F. Hsu, Y.-S. Lin, S.-Y. Chien, and W.-C. Chen, "Configuration through optimization for in-memory computing hardware and simulators," in *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, Nov. 2022, pp. 1–6. doi: 10.1109/SiPS55645.2022.9919208.
- [6] P. Wang *et al.*, "RC-NVM: Enabling symmetric row and column memory accesses for In-memory databases," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2018, pp. 518–530. doi: 10.1109/HPCA.2018.00051.
- [7] B. Ahn, J. Jang, H. Na, M. Seo, H. Son, and Y. H. Song, "AI accelerator embedded computational storage for large-scale DNN models," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2022, pp. 483–486. doi: 10.1109/AICAS4282.2022.9869991.
- [8] W. Kang, H. Zhang, and W. Zhao, "Spintronic memories: From memory to computing-in-memory," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, Jul. 2019, pp. 1–2. doi: 10.1109/NANOARCH47378.2019.181298.
- [9] R. Gauchi *et al.*, "Memory sizing of a scalable SRAM in-memory computing tile based architecture," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct. 2019, pp. 166–171. doi: 10.1109/VLSI-SoC.2019.8920373.
- [10] I. Pomeranz and S. M. Reddy, "Forward-looking fault simulation for improved static compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1262–1265, 2001, doi: 10.1109/43.952743.
- [11] C. E. Shannon, "Von Neumann's contributions to automata theory," *Bulletin of the American Mathematical Society*, vol. 64, no. 3, pp. 123–129, 1958, doi: 10.1090/S0002-9904-1958-10214-1.
- [12] M. Davis, "Emil Post's contributions to computer science," in *[1989] Proceedings. Fourth Annual Symposium on Logic in Computer Science*, 1989, pp. 134–136. doi: 10.1109/LICS.1989.39167.
- [13] L. Perri, "What's new in the 2022 Gartner hype cycle for emerging technologies," *Gartner*, 2022. <https://www.gartner.com/en/articles/what-s-new-in-the-2022-gartner-hype-cycle-for-emerging-technologies> (accessed Nov. 28, 2022).
- [14] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital system testing and testable design*. New York: Computer Science Press, 1998.
- [15] N. Takahashi, N. Ishiura, and S. Yajima, "Fault simulation for multiple faults by Boolean function manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 531–535, Apr. 1994, doi: 10.1109/43.275363.
- [16] M. Srivastava, S. K. Goyal, A. Saraswat, and G. Gangil, "Simulation models for different power system faults," in *2020 IEEE International Conference on Advances and Developments in Electrical and Electronics Engineering (ICADEE)*, Dec. 2020, pp. 1–6. doi: 10.1109/ICADEE51157.2020.9368915.
- [17] Menon and Chappell, "Deductive fault simulation with functional blocks," *IEEE Transactions on Computers*, vol. C-27, no. 8, pp. 689–695, Aug. 1978, doi: 10.1109/TC.1978.1675175.
- [18] Z. Navabi, *Digital system test and testable design*. Boston, MA: Springer US, 2011. doi: 10.1007/978-1-4419-7548-5.
- [19] I. Hahanov, S. Chumachenko, I. Iemelianov, V. Hahanov, L. Larchenko, and T. Daniyil, "Deductive qubit fault simulation," in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017, pp. 256–259. doi: 10.1109/CADSM.2017.7916129.
- [20] V. Hahanov, A. V. Hacimahmud, E. Litvinova, S. Chumachenko, and I. Hahanova, "Quantum deductive simulation for logic functions," in *2018 IEEE East-West Design & Test Symposium (EWDTS)*, Sep. 2018, pp. 1–7. doi: 10.1109/EWDTS.2018.8524619.
- [21] V. Hahanov, W. Gharibi, E. Litvinova, and S. Chumachenko, "Qubit-driven fault simulation," in *2019 IEEE Latin American Test Symposium (LATS)*, Mar. 2019, pp. 1–7. doi: 10.1109/LATW.2019.8704583.
- [22] W. Gharibi, D. Devadze, V. Hahanov, E. Litvinova, and I. Hahanov, "Qubit test synthesis processor for SoC logic," in *2019 IEEE East-West Design & Test Symposium (EWDTS)*, Sep. 2019, pp. 1–5. doi: 10.1109/EWDTS.2019.8884476.
- [23] D. B. Armstrong, "A deductive method for simulating faults in logic circuits," *IEEE Transactions on Computers*, vol. C-21, no. 5, pp. 464–471, May 1972, doi: 10.1109/T-C.1972.223542.
- [24] V. Hahanov *et al.*, "Vector-qubit models for SoC logic-structure testing and fault simulation," in *2021 IEEE 16th International Conference on the Experience of Designing and Application of CAD Systems (CADSM)*, Feb. 2021, pp. 24–28. doi: 10.1109/CADSM52681.2021.9385266.
- [25] V. I. Hahanov, S. M. Hyduke, W. Gharibi, E. I. Litvinova, S. V. Chumachenko, and I. V. Hahanova, "Quantum models and method

- for analysis and testing computing systems,” in *2014 11th International Conference on Information Technology: New Generations*, Apr. 2014, pp. 430–434. doi: 10.1109/ITNG.2014.125.
- [26] M. Karavay, V. Hahanov, E. Litvinova, H. Khakhanova, and I. Hahanova, “Qubit fault detection in SoC logic,” in *2019 IEEE East-West Design & Test Symposium (EWDTS)*, Sep. 2019, pp. 1–7. doi: 10.1109/EWDTS.2019.8884475.
- [27] V. Hahanov, *Cyber physical computing for IoT-driven services*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-54825-8.
- [28] V. Hahanov, I. Yemelyanov, V. Obrizan, and I. Hahanov, “‘Quantum’ diagnosis and simulation of SoC,” in *2015 XI International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, 2015, pp. 58–60.
- [29] V. Hahanov, E. Litvinova, O. Shevchenko, S. Chumachenko, H. Khakhanova, and I. Hahanov, “Vector models for modeling logic based on XOR-relations,” in *2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Feb. 2022, pp. 823–828. doi: 10.1109/TCSET55632.2022.9766894.
- [30] M. J. Biercuk and T. M. Stace, “Quantum computing’s achilles. Unavoidable errors and how to fix them,” in *IEEE Spectrum*, 2022, pp. 28–33.
- [31] R. Versluis and C. Hagen, “Quantum computers scale up: Constructing a universal quantum computer with a large number of qubits will be hard but not impossible,” *IEEE Spectrum*, vol. 57, no. 4, pp. 24–29, Apr. 2020. doi: 10.1109/MSPEC.2020.9055969.
- [32] M. Amaral, D. Chester, F. Fang, and K. Irwin, “Exploiting anyonic behavior of quasicrystals for topological quantum computing,” *Symmetry*, vol. 14, no. 9, Aug. 2022. doi: 10.3390/sym14091780.
- [33] G. Harutunyan, V. A. Vardanian, and Y. Zorian, “Minimal march tests for unlinked static faults in random access memories,” in *23rd IEEE VLSI Test Symposium (VTS’05)*, 2005, pp. 53–59. doi: 10.1109/VTS.2005.56.
- [34] Y. Zorian, A. Paschalis, D. Gizopoulos, and M. Psarakis, “Sequential fault modeling and test pattern generation for CMOS iterative logic arrays,” *IEEE Transactions on Computers*, vol. 49, no. 10, pp. 1083–1099, 2000. doi: 10.1109/12.888044.
- [35] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian, “RAM-based FPGAs: a test approach for the configurable logic,” in *Proceedings Design, Automation and Test in Europe*, 1998, pp. 82–88. doi: 10.1109/DATE.1998.655840.
- [36] U. Reinsalu, J. Raik, R. Ubar, and P. Ellervee, “Fast RTL fault simulation using decision diagrams and bitwise set operations,” in *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct. 2011, pp. 164–170. doi: 10.1109/DFT.2011.42.
- [37] R. Ubar, S. Devadze, J. Raik, and A. Jutman, “Fast fault simulation for extended class of faults in scan path circuits,” in *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications*, 2010, pp. 14–19. doi: 10.1109/DELTA.2010.32.
- [38] U. Reinsalu, J. Raik, and R. Ubar, “Register-transfer level deductive fault simulation using decision diagrams,” in *2010 12th Biennial Baltic Electronics Conference*, Oct. 2010, pp. 193–196. doi: 10.1109/BEC.2010.5631842.
- [39] I. Pomeranz and S. M. Reddy, “Unspecified transition faults: A transition fault model for at-speed fault simulation and test generation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 137–146, Jan. 2008. doi: 10.1109/TCAD.2007.907000.
- [40] I. Pomeranz and S. M. Reddy, “Test data compression based on output dependence,” in *2003 Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 1186–1187. doi: 10.1109/DATE.2003.1253793.
- [41] J. Joe, N. Mukherjee, I. Pomeranz, and J. Rajski, “Fast test generation for structurally similar circuits,” in *2022 IEEE 40th VLSI Test Symposium (VTS)*, Apr. 2022, pp. 1–7. doi: 10.1109/VTS52500.2021.9794232.
- [42] I. Pomeranz, “Positive and negative extra clocking of LFSR seeds for reduced numbers of stored tests,” in *2021 IEEE 30th Asian Test Symposium (ATS)*, Nov. 2021, pp. 109–114. doi: 10.1109/ATS52891.2021.00031.
- [43] I. Pomeranz and M. E. Amyeen, “Hybrid pass/fail and full fail data for reduced fail data volume,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1711–1720, Aug. 2021. doi: 10.1109/TCAD.2020.3025091.
- [44] I. Pomeranz and S. M. Reddy, “A synthesis procedure for flexible logic functions,” in *Proceedings Design, Automation and Test in Europe*, 2002, pp. 973–974. doi: 10.1109/DATE.1998.655995.
- [45] I. Pomeranz, L. N. Reddy, and S. M. Reddy, “COMPACTEST: a method to generate compact test sets for combinational circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1040–1049, Jul. 1993. doi: 10.1109/43.238040.
- [46] I. Pomeranz and Y. Zorian, “Fault isolation for nonisolated blocks,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 12, pp. 1412–1415, Dec. 2004. doi: 10.1109/TVLSI.2004.837994.
- [47] S. Temma, M. Sugii, and H. Matsuno, “The document similarity index based on the Jaccard distance for mail filtering,” in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, Jun. 2019, pp. 1–4. doi: 10.1109/ITC-CSCC.2019.8793419.
- [48] P. Jaccard, “Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines,” *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, no. 140, pp. 241–272, 1901. doi: 10.5169/seals-266440.
- [49] “Functional verification.”
- [50] “Smart everything starts with silicon.”
- [51] N. Vinod *et al.*, “Performance evaluation of LUTs in FPGA in different circuit topologies,” in *2020 International Conference on Communication and Signal Processing (ICCSPP)*, Jul. 2020, pp. 1511–1515. doi: 10.1109/ICCSPP48568.2020.9182074.




## BIOGRAPHIES OF AUTHORS

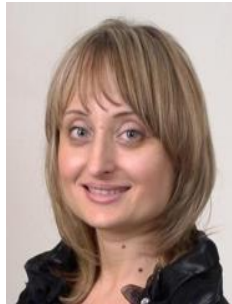





**Wajeb Gharibi**    is a professor of Computer Science and Electrical Engineering, University of Missouri Kansas City, USA. His research and development fields include the design and testing of computers, quantum memory-driven computing, cyber-physical and cyber-social computing, pattern recognition, machine learning computing, and digital smart cyber university. Cloud-driven traffic control. He can be contacted at gharibiw2002@yahoo.com.








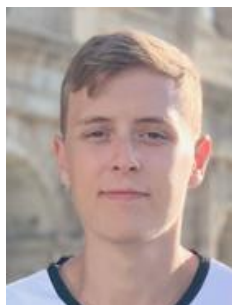
**Vladimir Hahanov**    was born in USSR in 1953. He is a Doctor of Science, Professor of Computer Engineering Faculty, Design Automation Department, Kharkiv National University of Radio Electronics, Ukraine. His research and development fields include design and test of computers, test generation and fault simulation for soc, quantum memory-driven computing, cyber-physical, cyber social computing, pattern recognition and machine learning computing, digital smart cyber university, and cloud-driven traffic control. He has supervised 4 Doctors of Science and 35 Ph.Ds. He has been the General Chair of the IEEE East-West Design & Test Symposium for 20 years since 2003. He is also the author of 650+ publications and 25 textbooks, 5 patents, and 188 Scopus-indexed papers: with 467 citations by 315 documents. Prof. Hahanov has been IEEE Senior Member since 2010, IEEE Computer Society Golden Core Member, SAE member, and IFAC member. He can be contacted at hahanov@icloud.com.






**Svetlana Chumachenko**    was born in USSR in 1969. She is a Doctor of Technical Sciences, Professor, and Head of the Design Automation Department, Kharkiv National University of Radio Electronics, Ukraine. Her research and development fields include Mathematics, Computer Engineering, and Smart Cyber University. Her international activities include the fundamental research within agreement cooperation on scientific and technical “Strategic partnership” with the firm Aldec Inc. (USA) in 2000, 2005; SEIDA BAITSE “Baltic Academic IT Security Exchange”, Blekinge Institute of Technology, Sweden in 2011–2012; international project “Curricula Development for New Specialization: Master of Engineering in Microsystems Design 530785-TEMPUS-1-2012-1-PL-TEMPUS-JPCR” Priority – Curricula Reform in 2012 to 2016. She can be contacted at svetachumachenko@icloud.com.






**Eugenia Litvinova**    was born in Ukraine in 1962. She is a Doctor of Science and a Professor of Computer Engineering Faculty, Design Automation Department, Kharkiv National University of Radio Electronics, Ukraine. Her research and development fields include the design and testing of computers and quantum computing. Her international activities include being a staff member of the Tempus Project No. 530785-TEMPUS-1-2012-PL-TEMPUS-JPCR Curricula Development for New Specialization: Master of Engineering in Microsystems Design and also a member of the organizing committee of IEEE East-West Design Test Symposium from 2007 to present. She can be contacted at litvinova\_eugenia@icloud.com.



**Ivan Hahanov**    was born in Ukraine in 1997. He is Ph.D. Student of Computer Engineering, Kharkiv National University of Radio Electronics, Ukraine. Ivan Hahanov is an Experienced Coursera-certified ML specialist and ex-Mobile developer. His research and development fields include computer vision, machine learning, deep learning, data analysis, NLP, MLOps, cloud, and big data. Ivan Hahanov is the author of 31 publications, Scopus h-Index: 3, 31 citations by 22 documents. He can be contacted at ivanhahanov@icloud.com.



**Irina Hahanova**    was born in Ukraine. In 1994, she graduated from the Kharkiv National University of Radio Electronics, majoring in Computers, Complexes, and Networks (qualification - systems engineer). In 1998, she defended her thesis for the degree of Candidate of Technical Sciences, and in 2008 she defended her doctoral thesis on the specialty 05.13.05 - Computer Systems and Components. Her dissertation topic is "Models and methods of system design of computational structures on crystals for digital signal processing." She can be contacted at irina.hahanova@nure.ua.