

Vector synthesis of fault testing map for logic

Vladimir Hahanov¹, Wajeb Gharibi², Svetlana Chumachenko¹, Eugenia Litvinova¹

¹Design Automation Department, Computer Engineering Faculty, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine

²Department of Computer Science and Electrical Engineering, University of Missouri, Kansas City, United States

Article Info

Article history:

Received Mar 2, 2024

Revised Jun 24, 2024

Accepted Jul 7, 2024

Keywords:

Fault simulation mechanism

Fault testing map

Fault truth table

In-memory computing

Logic vector computing

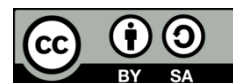
Smart data structures

Test truth table

ABSTRACT

Vector synthesis of fault testing (simulation) map for logic is proposed, which without simulation allows to determine of all faults detected on test sets, as well as determining test sets to detect specified faults. For synthesis, a superposition of smart data structures is used, containing: a deductive matrix D , as a derivative of the logical vector L , test truth table T , and fault truth table F . The deductive matrix is seen as the gene of functionality and base of fault simulation mechanism to solve all the problems of testing and verification. In the matrix synthesis, an axiom is used: all the mentioned tables are identical in shape to each other and always interact with each other convolutionally $T \oplus L \oplus F = 0$. A universal deductive reversing converter “test-faults” and “faults-test” for logical functionalities of any dimension is proposed. Converter functions: fault simulation on test sets $T \rightarrow F$ and synthesis of test sets $F \rightarrow T$ to detect the specified faults. The converter can be used as a test generation and fault simulation service for IP-core system-on-chip (SoC) under the IEEE 1500 SECT standard. Based on the deductive matrix, a fault testing map for logic is built, where each test set is matched with the logic-detected faults of the input lines.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Vladimir Hahanov

Design Automation Department, Computer Engineering Faculty, Kharkiv National University of Radio Electronics

Kharkiv, Ukraine

Email: hahanov@icloud.com

1. INTRODUCTION

System-on-chip (SoC) IP-core testing is dealt with by thousands of specialists and hundreds of companies that find reasonable technological and software solutions to create a product yield in a reasonable time-to-market [1]–[4]. At the same time, companies should pay more attention to mathematics and data structures, which could be better in almost most industrial systems for designing and testing digital devices. The market design metric “quick solution is better than correct one” leads any system with hundreds of software and hardware patches to a state inoperable for the market over a long time [1], [5]–[7]. The way out: the development of mathematically smart, as a rule, long-lived data structures invariant to the rapid change in programming technologies and the element base of digital systems [1], [8]–[10]. On such well-thought-out data structures, very compact, simple algorithms for their analysis are obtained [1], [3], [11]–[14]. There is a design pattern: if you spend 90% of your time on the mathematical culture of smart data structures, the time for creating simple algorithms for their processing will not be more than 10% [13], [15]–[20].

The Intelligent Computing Technology Metric [21]–[30] offers the following time and energy saving solutions. First is read-write transactions on memory [16]–[21], [26]. Second is in-memory synthesis of a simple logical processor [10], [11], [28]. Third is using RISC-V architecture instructions to process data in memory [21], [24], [31]. Fourth is synthesis of smart data structures for information processing without an

algorithm [22], [23], [27], [29]. Finally, it is Neumann architecture in-memory to control data processing [21], [25], [30].

In-memory computing is gradually capturing new areas of the information technology (IT) industry due to the absence of iterations between memory and the central processor, which significantly saves energy in the intelligent processing of big data [21]–[30]. In addition, redundancy memory has reached the processor's logical command speed [21], [24], [26], [30], forcing companies to leverage memory services to organize computing processes. Many publications consider memory as a convenient tool for implementing big data processing algorithms on a limited set of instructions for deterministic and probabilistic computing, including machine learning and artificial neural networks [22]–[25], [27], [28]. In addition, only in-memory read-write transactions can be leveraged to build big data processing architectures free from processor instructions [16]–[21]. The synthesis of smart data structures makes it possible to code computational algorithms based on in-memory read-write transactions [1], [16]–[19], [30] with a speed of 1 ns or less.

Statement of the fault simulation problem. Fault simulation is essential for Automatic Test Pattern Generation (ATPG) [3]–[5], [10]–[13], diagnosis [10]–[12], [16], [17], and fault and test grading [16], [17], [32]–[35]. Fault simulation metrics are speed, memory, modeling functional blocks and delay capability, sequential circuits, multi-valued fault simulation to handle unknown X, and high-impedance Z [16], [17], [32]–[35]. In addition to the circuit model, a fault simulator usually needs stimuli and expected responses (required for true-value simulation), a fault model, and a fault list. Concurrent fault simulation [33], [34] is an event-driven simulation that involves good/bad events together. The simulated part of faulty circuits differs from that of good circuits. Memory management is complex. Practically all industrial fault simulation systems have unpredictable sizes of fault lists and unpredictable sizes of data structures. All six primary simulation engines use a processor with a high-power consumption level. The Electronic Design Automation (EDA) market needs simple and efficient fault simulation mechanisms to verify tests [1], [18], [19]. A metric comparison of fault simulation mechanisms existing in industrial systems is given in Table 1. Vector fault simulation is also evaluated in this metric, for which all metric points, except for one here, are in black.

Table 1. Fault simulation techniques: metrics comparison

Fault simulation technique	Complexity	Memory	Data structure	Level	Delay	Speed	Fault model	Multi-valued
Serial fault simulation [34]	$n \times n^3$	Predictable	Add fault model, fault list	Gate, system	No problem	Slowest	Any	Easy
Parallel fault simulation [3]	$\frac{1}{w} \times n^3$	Predictable	Register Memory	Gate	Not capable	Middle	Logic	Difficult
Deductive fault simulation [32]	n^2	Unpredictable	Deductive formulas	Gate	Not capable	Middle	Any	Difficult
Concurrent fault simulation [33]	$\frac{1}{3} \times n^2$	Unpredictable	Add fault model, fault list	Gate, RTL	Capable	Faster	Logic	Easy
PPSFP – Parallel pattern single fault propagation [10]	$\frac{1}{w} \times n^3$	Unpredictable	Add fault model, fault list	Gate	Capable	Middle	Logic	Easy
Differential fault simulation [35]	$\frac{1}{2} \times n^2$	Unpredictable	Add fault model, fault list	Gate, RTL	Not capable	Middle	Any	Difficult
Vector fault simulation [18]	$\frac{1}{2} \times \frac{1}{3} \times n^2$	Predictable	No, as true-value simulation	Gate, RTL, System	Not capable	Faster	Logic	Capable

The goal is to significantly reduce the latency and energy consumption of in-memory computing for fault simulation of Boolean functionality by modeling a deductive matrix via smart data structures on a logic vector. The objective of this paper is to i) develop smart (connected) data structures based on a logical vector, including a deductive matrix and two truth tables: tests and faults, ii) development of an algorithm for simulation faults on a test set, iii) create of an algorithm for generating test sets for given faults, iv) synthesis of a reversible converter of test sets into detected faults and vice versa, detected faults into test sets, v) synthesis of fault testing map for logic using a deductive matrix, and vi) verification of data structures and algorithms for their analysis.

A new scientific direction of vector logical computing (VLC) is proposed. This is processor-free computation in memory based on read-write transactions on logical vectors. The new vector logic computing metrics for fault modeling and simulation are i) free-from-processor data processing means energy-saving computing technology, ii) the absence of the classical von Neumann architecture means no iterations between the data bus between memory and the arithmetic logic unit (ALU), significantly reducing data processing time, iii) read-write transactions instead of a robust processor instruction set that allows you to organize the

computational process on any memory device, iv) logical vectors assume their simple placement in memory without time-consuming synthesis into a technologically permitted system of elements, v) an address is an attribute of memory and truth tables and a fault as address simulation mechanism, vi) smart data structures reduce the algorithm for their analysis to linear computational complexity, vii) all digital functionality testing tasks are solved using a logical vector and its derivatives, and viii) the simulation of tests and faults on the truth table addresses does not include the simulation algorithm. Everything set out in the metric of vector-logical computing, as well as models, methods, algorithms, and mechanisms for fault as address simulation, is new and has no global analogs. The advantage of the proposed study is that it reduces models and algorithms to simple engineering mechanisms.

2. DEDUCTIVE MATRIX SYNTHESIS

The truth table is an ideal model for solving all combinatorial problems, including test synthesis and fault simulation [17]. The logical vector is the most compact universal model of a logical element. D-matrix is a genome for solving all problems in the field of design and testing. These three models comprise smart data structures for organizing any in-memory testing SoC IP-core. The properties of the deductive matrix obtained from the logical vector for solving the problems of synthesis and analysis of the test by simulation faults are considered. Smart data structures make it possible to solve the problem of transporting input faults to the output without simulation when a binary set is put to the inputs of a logic element. The problem of synthesis of a minimum test for input faults of a component is also solved by trivial analysis of the deductive matrix. The essence of these methods is to build smart data structures based on a logical vector (truth table), by interacting the logical vector “with itself,” a quadratic matrix of the activity of the logical element is built. Recoding the coordinates of this matrix by binary-decimal addresses leads to a deductive matrix, which initially solves the problems of test generation and fault simulation efficiently. The computational complexity of solving these problems is determined by a quadratic estimate of the length of the logical vector $C=2^n \times 2^n$. To create a deductive matrix, follow five steps of the algorithm as shown in Figure 1.

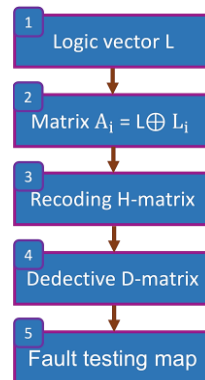


Figure 1. Algorithm for creating deductive matrix

- i. Initial data: the logical L-vector of functionality from n-variables is written horizontally and vertically in the active A-matrix in Figure 2.
- ii. The activity matrix A, consisting of logical vectors and their inversions, is created based on the given logical vector. Construction of A-activity table rows as shown in Figure 2 carried out according to the rule: $A_i = L \oplus L_i$. Here Q_i , the state of the i-bit of the logical (vertical) L-vector. Put, if the coordinate of the vertical L-vector is equal to 1, then the inversion of the logical vector $A_i = \bar{L}$ is written to the corresponding row of the A-matrix. Otherwise, the logical vector $A_i = L$ is written to the row. Empty cells in the matrices denote zero coordinates to reduce the load on vision. The activity A-matrix is always symmetrical in relation to the main diagonal. A deductive matrix is an active matrix, bits of which are ordered by binary-decimal addresses via an H-matrix.
- iii. The bit-recoding matrix H is needed to build a deductive matrix quickly. The H-matrix is the same for all logical functionalities from n variables. The recursive mechanism of H-matrix synthesis in Figure 3 uses the operations sequentially to calculate the coordinates of the four quadrants: 1) $H_{i+1}^1 = H_i$; 2) $H_{i+1}^2 = 2^n + H_i$; 3) $H_{i+1}^3 = H_{i+1}^2$; $H_{i+1}^4 = H_{i+1}^1$.

- iv. Creating of deductive D-matrix by the formula: $D_j = L_{H_{ij}}, j = \overline{1, 2^n}$, which recodes the bits of the A-activity matrix according to the H-matrix recoding coordinates addresses (obtained in step 3 of the algorithm). Thus, the D-matrix is obtained based on the execution of the operator $D = (L \oplus L_i)_H$, obtained because of the superposition of the operators: $A = L \oplus L_i$ and $D = A_H$. D-matrix is a genome for solving all problems in the field of design and testing.
- v. A synthesis of a logical functionality testing map is carried out using a deductive matrix. Section 5 of this paper will describe this procedure.

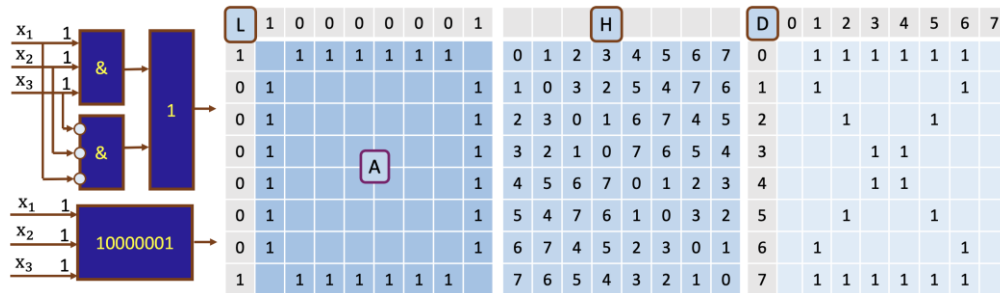


Figure 2. Procedure for creating a deductive matrix

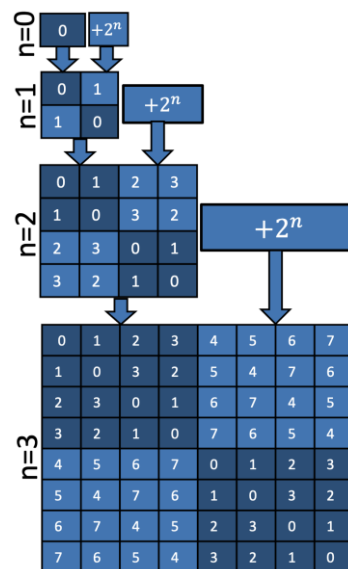


Figure 3. Recursive scheme for obtaining H-matrix of recoding

3. ANALYSIS OF DEDUCTIVE MATRICES

It is straightforward to create deductive matrices of logical functionalities in n -variables. However, it remains to clarify the deductive matrix's beneficial properties for solving fault simulation problems and testing the generation of logical functionalities. Deductive matrices of three essential elements from three variables have the form in Figure 4.

The first helpful property of the deductive matrix is that without simulation, it determines all input faults detected on the test set. Simulation formula: Based on the binary test set, a row of the D-matrix is selected, the 1-unit values of which activate the 1-unit coordinates of the upper fault TT, which are additionally determined by the inverse values of the coordinates of the selected column of the truth table equal to the test set. Following the simulation formula, we determine which faults are detected on test set 101 of all three elements. The first matrix gives the following result: stuck-at-1 is detected on the second input $X_2=1$. The second matrix forms the result of the multiple fault check: $X_1, X_3 = 00$. The third matrix forms the following checked stuck-at-faults: $X_1 = 0, X_2 = 1, X_3 = 0$, multiple fault $X_1, X_2, X_3 = 010$. Thus, the

deductive matrix without simulation detects faults of any combination on the binary input set, which are described by 1-unit coordinates in the columns of the fault TT. All interacting components, such as tests, faults, and logic, are defined on the same truth table (logical vector), the derivative of which is the deductive matrix.

X_1	X_2	X_3	D (L=00000001)								X_1	X_2	X_3	D (L=01111111)								X_1	X_2	X_3	D (L=01101001)							
		X_1	0	1	0	1	0	1	0	1			X_1	0	1	0	1	0	1	0	1			X_1	0	1	0	1	0	1	0	1
		X_2	0	0	1	1	0	0	1	1			X_2	0	0	1	1	0	0	1	1			X_2	0	0	1	1	0	0	1	1
		X_3	0	0	0	0	1	1	1	1			X_3	0	0	0	0	1	1	1	1			X_3	0	0	0	0	1	1	1	1
0	0	0								1	0	0	0	0	1	1	1	1	1	1	1	0	0	0		1	1		1			1
1	0	0								1	1	0	0		1							1	0	0		1	1		1			1
0	1	0								1	0	1	0			1						0	1	0		1	1		1			1
1	1	0								1	1	1	0			1						1	1	0		1	1		1			1
0	0	1								1	0	0	1									0	0	1		1	1		1			1
1	0	1								1	1	0	1									1	0	1		1	1		1			1
0	1	1								1	0	1	1									0	1	1		1	1		1			1
1	1	1	0	1	1	1	1	1	1	1	1	1	1									1	1	1		1	1		1			1
Derivatives →			X'_1	X'_2	X'_{12}	X'_3	X'_{13}	X'_{23}	X'_{123}		Derivatives →			X'_1	X'_2	X'_{12}	X'_3	X'_{13}	X'_{23}	X'_{123}		Derivatives →			X'_1	X'_2	X'_{12}	X'_3	X'_{13}	X'_{23}	X'_{123}	

Figure 4. Deductive matrices of three basic logical functions

4. DEDUCTIVE REVERSIBLE CONVERTER

Next, a deductive reversible converter is proposed that, without simulation, allows you to determine all the faults detected on test sets (define test sets for detected given faults), thanks to the superposition of smart data structures in memory [17], including two truth tables: test TT, faults TT and deductive D-matrix. Figure 5 shows an example of the synthesis of a deductive matrix and its leverage for fault simulation and test generation on the same data structures. Empty cells in the active and deductive matrix and faults TT indicate zero coordinates.

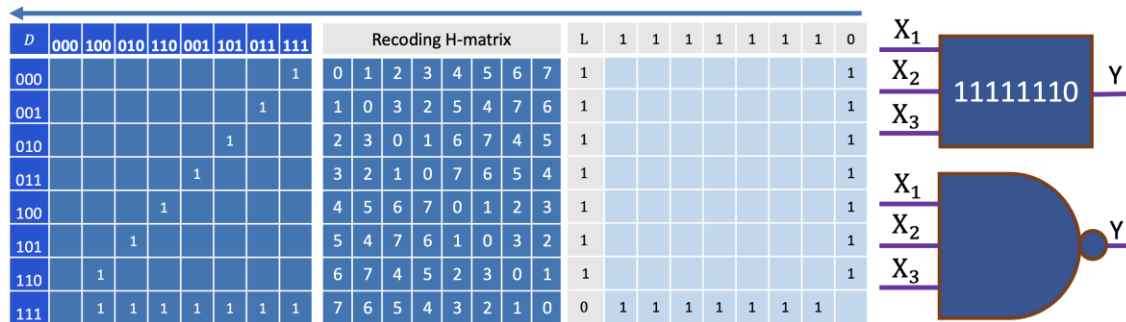


Figure 5. Synthesis of the deductive matrix of a logical element

Using the deductive matrix, one can quickly generate a test for stuck-at-faults of the input variables (columns 1, 2, 4), which has the form of $T = (110, 111, 101, 111, 011, 111) = (110, 101, 011, 111)$. The test is obtained by projecting each column (1, 2, 4) with faults to all 1-unit coordinates of the D-matrix, the projection of which onto the test TT rows form the minimum test. Geometrically, the algorithm is represented by the angle “ \downarrow ” of two arrows that define test sets by the deductive matrix: \downarrow – from each chosen column of faults TT, descend to the intersection with the matrix 1-units, \leftarrow – then from the matrix 1-units we move to the left until the intersection with the rows of the test TT.

As for the simulation of faults for the $L=01111111$ element: let us have 4 test sets: 110, 101, 011, 111. When simulating manually, we will carry out such a “ $\rightarrow\uparrow$ ” movement along the matrix from the test set to the right until it intersects with 1-units and then up from them to the intersection with the columns of faults TT, where each 1-unit of the column is converted into a fault, the sign of which is determined by the inversion of the state of the input variable on the test set. The results of the simulation of faults on test sets are as follows: 110 – $X_1 = 1$, 101 – $X_2 = 1$, 011 – $X_1 = 1$, 111: $X_1 = 0$, $X_2 = 0$, $X_3 = 0$, and all their four possible combinations.

An innovative solution to design and test problems is practically proposed as a universal deductive reversible converter “test faults” and “faults \rightarrow test” based on the deductive matrix in Figure 6 for logical functionalities of any dimension. The functions of the converter are i) fault simulation on test sets $F=D(T)$ and ii) generating test sets that detect given faults $T=D(F)$. On the converter, it is possible to perform fault diagnosis of logical devices on given test sets and verify the test to detect a given set of faults in a logic element in percent.

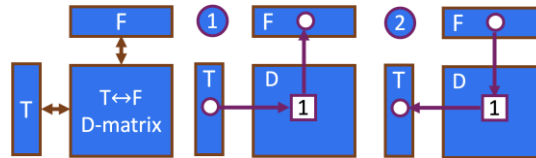


Figure 6. Deductive reversible test-fault converter and his two functions

To create such a converter from three smart (connected) components, it is necessary to generate the rows A-activity matrix leveraging the coordinate-vector operation $A_i = L \oplus L_i$ based on the logical L-vector and transform it into a deductive D-matrix using the recoding matrix of coordinates addresses $D = A_H$ as shown in Figure 7. The recoding matrix is the universal key for obtaining the deductive matrix of any n-input element. The deductive matrix is a genome for solving all the problems of testing digital functional elements of any complexity. The formula for obtaining the D-matrix is $D = (L \oplus L_i)_H$.

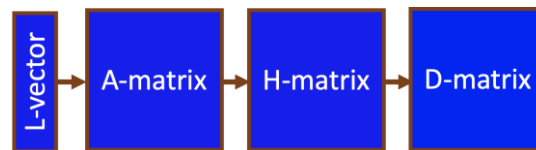


Figure 7. Scheme for obtaining a deductive matrix

The fee for creating such services is a sufficiently large memory required to store the deductive matrix and two truth tables $M = 2^n \times 2^n + 2n \times 2^n$, n – the number of input variables of the functional logic element. If we talk about the entire fault simulation mechanism (FSM) in Figure 8, then its implementation also requires memory for the synthesis of the deductive matrix: $M = 2 \times (2^n \times 2^n) + 2^n$. The memory estimation for data storage (logical vector, two truth tables, and three quadratic matrices) to implement the entire FSM is $M = 3(2^n \times 2^n) + 2^n + 2n \times 2^n$.

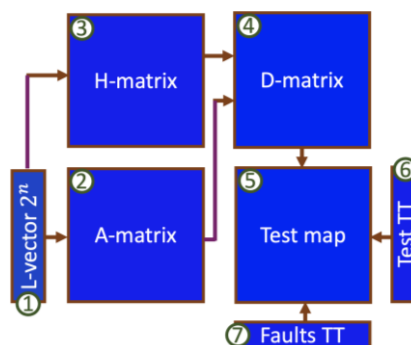


Figure 8. The structure of the fault simulation mechanism

Let us talk about the novelty of the research. Smart data structures based on the logical L-vector of the element are described here, which allows the creation of a deductive matrix (blocks 1 to 4) and leverage it

as a reverse in-memory converter for fault simulation and test synthesis (blocks 4 to 6) based on truth tables: Test TT, Faults TT and deductive D-matrices, which form a common topological space used to describe FSM. This mechanism contains four procedures for sequentially creating four matrices based on a logical L-vector: A-matrix – H-matrix – D-matrix – S-matrix (simulation). The last matrix is also called the fault testing map (FT-map). In this scheme, the essence of the mechanism for fault simulation of logical elements of any dimension. It could not be easier.

The formula of D-matrix synthesis [17], as presented in Figure 8, are as follows. According to the logical L-vector of functionality (block 1) a quadratic A-matrix of activity (block 2) is created, where each row is a logical vector or its or its inversion $A_i = L \oplus L_i$ depending on the state of the i-cell of the L-vector and, then all the coordinates of the resulting A-matrix are ordered by the indices of the quadratic recoding H-matrix (block 3) to obtain a deductive D-matrix (block 4 and 5), which is a reverse converter of the test set into detected faults and vice versa, where the 1-unit values of the D-matrix coordinate define a one-to-one correspondence between the rows (test sets) of test TT (block 6) and columns (combination of faults) faults TT (block 7). Formula for fault simulation: from the selected test set, move to the right along the term of the deductive matrix until it intersects with 1-coordinate, then move up from each 1-unit until it intersects with the columns of faults TT, where the inverse states of the input variables of the test set further define the 1-unit coordinates. Formula for test synthesis: from selected in faults TT of the column as combinations of faults specified by 1-units, the movement is carried out down the deductive matrix until it meets the 1-units of the D-matrix, after which it moves from 1-units to the left until it meets the test sets of Test TT, which always detect combinations of faults that are inverse to the states of the variables on the test set.

The technological novelty of the fault simulation mechanism lies in placing all components of smart data structures (6 components) in any memory that can be processed by a limited set of logical commands in the limit read–write transactions. The synthesis and operation of the converter do not require a powerful system of CPU instructions, which makes it economical in terms of energy and time for fault simulation (test generation) since a read-write transaction on emergence memory is completed in 1 ns.

5. SYNTHESIS FAULT TESTING MAP

The goal is to move from a deductive matrix to an explicit assignment of all faults to be detected on test input sets. To do this, it is necessary to define all 1-coordinates of the deductive matrix with fault vectors detected on the test set by constructing a projection of the 1-coordinate of the matrix onto the vectors of the test and fault truth tables. In other words, it is necessary to record the faults detected on the test set in the 1-coordinates of the deductive matrix. To do this, superpositions of two truth tables are performed: tests and faults based on a deductive matrix.

The examples of processing logical functionalities as shown in Figure 9 are switch circuits and adder circuits. These circuits have a known structure, but only a logical vector synthesizes the test-fault converter. Obtaining a fault testing map for these two functionalities is accompanied by three operations presented from left to right in Figure 10: i) obtaining an activity matrix by manipulating a logical vector, ii) synthesizing a deductive matrix based on applying bit transcoding matrix activity to the matrix, and iii) constructing a test-fault map by superposition of test sets with fault table codes at 1-values of the coordinates of the deductive matrix.

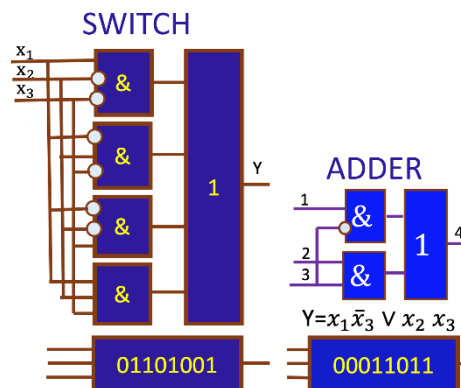


Figure 9. Logic circuits considered as elements

Test-map									D-matrix									H-matrix								Logic vector											
TT	000	001	010	011	100	101	110	111	TT	000	001	010	011	100	101	110	111													L	0	1	1	0	1	0	1
000	..1	..1	..1	..1	..1	..1	..1	..1	000	1	1	1	1	1	1	1	1	0	1	2	3	4	5	6	7	0	0	1	1	1	1	1	1	1	1	1	
001	..0	..1	..1	..1	..1	..1	..1	..1	001	1	1	1	1	1	1	1	1	1	0	3	2	5	4	7	6	1	1	1	1	1	1	1	1	1	1		
010	..1	..0	..1	..1	..1	..1	..1	..1	010	1	1	1	1	1	1	1	1	2	3	0	1	6	7	4	5	1	1	1	1	1	1	1	1	1	1		
011	..0	..0	..1	..1	..1	..1	..1	..1	011	1	1	1	1	1	1	1	1	3	2	1	0	7	6	5	4	0	1	1	1	1	1	1	1	1	1		
100	..1	..1	..0	..0	..0	..0	..0	..0	100	1	1	1	1	1	1	1	1	4	5	6	7	0	1	2	3	1	1	1	1	1	1	1	1	1	1		
101	..0	..1	..0	..0	..0	..0	..0	..0	101	1	1	1	1	1	1	1	1	5	4	7	6	1	0	3	2	0	1	1	1	1	1	1	1	1	1		
110	..1	..0	..0	..0	..0	..0	..0	..0	110	1	1	1	1	1	1	1	1	6	7	4	5	2	3	0	1	0	1	1	1	1	1	1	1	1	1		
111	..0	..0	..0	..0	..0	..0	..0	..0	111	1	1	1	1	1	1	1	1	7	6	5	4	3	2	1	0	1	1	1	1	1	1	1	1	1	1		

Test-map									T-matrix									F-matrix								L-matrix											
TT	000	001	010	011	100	101	110	111	TT	000	001	010	011	100	101	110	111													Y	0	0	0	1	1	0	1
000			..1	..1	..1	..1	..1	..1	000			1	1	1	1	1	1	0	1	2	3	4	5	6	7	0	0	1	1	1	1	1	1	1	1		
001		..1			..1	..1	..1	..1	001		1			1	1	1	1	1	0	3	2	5	4	7	6	0	0	1	1	1	1	1	1	1			
010	..1			..1	..1	..1	..1	..1	010	1			1	1	1	1	1	2	3	0	1	6	7	4	5	0	0	1	1	1	1	1	1	1			
011	..0	..0	..0		..1	..1	..1	..1	011	1	1	1	1	1	1	1	1	3	2	1	0	7	6	5	4	1	1	1	1	1	1	1	1	1			
100	..1			..0	..1	..1	..1	..1	100	1			1	1	1	1	1	4	5	6	7	0	1	2	3	1	1	1	1	1	1	1	1	1			
101	..0	..1	..1	..1	..1	..1	..1	..1	101	1	1	1	1	1	1	1	1	5	4	7	6	1	0	3	2	0	0	1	1	1	1	1	1	1			
110			..1	..0	..0	..0	..0	..0	110			1	1	1	1	1	1	6	7	4	5	2	3	0	1	1	1	1	1	1	1	1	1	1			
111		..0			..0	..0	..0	..0	111		1			1	1	1	1	7	6	5	4	3	2	1	0	1	1	1	1	1	1	1	1	1			

Figure 10. Build fault testing map for two circuits treated as elements

The formula for constructing a fault testing map from a deductive matrix: each 1-coordinate of the deductive matrix is determined by the inverse values of the coordinates of the test (left) set for all 1-coordinates of the (upper) vector of the fault table. The obtained vector of the faults to be tested in place of the unit coordinates of the deductive matrix is determined by the alphabet (0,1, “.”), where the third character indicates the absence of a fault check at this input coordinate. Each unit coordinate of the deductive matrix denotes the meeting point of the test set with the fault vector, which forms, at this point, the vector of input faults checked on the test set. Thus, the test-fault card solves the issues of fault simulation, fault diagnosis, and test generation for logical functionality of any complexity without processing. Given that any process or phenomenon can be easily digitized and turned into a logical function, this map can serve as a tool for identifying errors in a business process, a social process, or a critical process, including transport, nuclear energy, astronautics, and urban infrastructure. Figure 11 shows the symmetrical circuit and its representation as a function. The synthesis of the fault testing map in Figure 12 of the functionality specified by the logical vector 11000011. The deductive D-matrix is a genome for solving all problems of testing digital functional elements of any complexity. The formula for obtaining a D-matrix is $D=(L\oplus L_i)_H$. D-matrix functions are fault simulation on test sets $F=D(T)$ and definition of test sets for fault detection $T=D(F)$. These two functions are explicitly shown on the fault testing map.

The formula for the synthesis of the D-matrix: a quadratic A-matrix of activity is constructed from the logical L-vector of functionality, where each line is a logical vector or its or an inversion $A_i=L\oplus L_i$ depending on the state of the i-cell of the L-vector, then all the coordinates of the resulting matrix are ordered according to the indices of the quadratic H-matrix (secret key) of bit recoding to obtain a deductive D-matrix, which is a reversible converter of the test set into testable faults and vice versa, where the unit values of the coordinates of the D-matrix build a functional correspondence between the truth table test sets and the faults being tested fault truth table. This fault simulation mechanism has no global analogs regarding manufacturability and efficiency. H-matrix bit recoding brings the unordered bits of the A-matrix activity to the standard order of the addresses of the D-matrix rows and columns defined by the binary addresses of the test and fault truth tables.

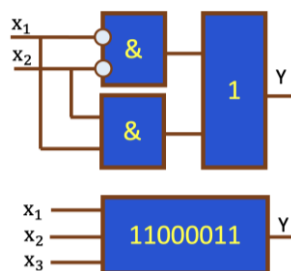


Figure 11. Structural and functional circuits with symmetry

Test-map								D-matrix								H-matrix								Logic vector							
000 001 010 011 100 101 110 111								000 001 010 011 100 101 110 111																1 1 0 0 0 0 1 1							
000		.1.	.11	1..	1.1			000		1	1	1	1			0	1	2	3	4	5	6	7	1		1	1	1	1		
001		.1.	.10	1..	1.0			001		1	1	1	1			1	0	3	2	5	4	7	6	1		1	1	1	1		
010		.0.	.01	1..	1.1			010		1	1	1	1			2	3	0	1	6	7	4	5	0	1	1		1	1		
011		.0.	.00	1..	1.0			011		1	1	1	1			3	2	1	0	7	6	5	4	0	1	1		1	1		
100		.1.	.11	0..	0.1			100		1	1	1	1			4	5	6	7	0	1	2	3	0	1	1		1	1		
101		.1.	.10	0..	0.0			101		1	1	1	1			5	4	7	6	1	0	3	2	0	1	1		1	1		
110		.0.	.01	0..	0.1			110		1	1	1	1			6	7	4	5	2	3	0	1	1		1	1	1	1		
111		.0.	.00	0..	0.0			111		1	1	1	1			7	6	5	4	3	2	1	0	1		1	1	1	1		

Figure 12. Fault testing map synthesis for symmetric logical functionality

The superposition of the D-matrix and the two truth tables (test and fault) creates a test-set and fault-set coordinate system (T, F) for all 1-units of the D-matrix. The test map synthesis formula is determined by the superposition of a pair of coordinates for each 1-unit cell of the D-matrix to determine the 1-coordinates of the fault-set with the inverse values of the corresponding bits of the test-set according to the simple rule: If $F_i = 1$, then $D_i = \bar{T}_i$. The truth table of this analytical coordinate converter (T, L) to F is shown in Table 2.

Table 2. Identifying input line faults

T_i	L_i	F_i
0	0	.
0	1	.
1	0	1
1	1	0

Similarly, you can create an alternative test map by placing test vectors on the unit coordinates of the D-matrix, which will detect faults defined in the fault truth table. Thus, it can be argued that the D-matrix is the genome for solving all the problems of functional testing. The matrix's unit coordinates indicate critical functionality areas that test benches must detect. It can be argued that the fault truth table of functionality is a universal source form from which all kinds of faults and functional coverage are extracted. A test is built for malfunctions if the goal is to determine the location, cause, and types of defects in the circuit or model. The functional coverage test aims to verify the functionality defined in the specification. There may be errors in the synthesis stages when one functionality is synthesized. To do this, it is enough to introduce fault into one bit of the logical vector. Detection of such faults is possible only on an exhaustive or complete test. A compromise or cheaper option can be a test to cover the critical points of functionality determined by 1-values of the coordinates of the deductive matrix. Automatic test-bench synthesis can substantially reduce the time-to-market of a digital product with valid Yield parameters.

The computational complexity of algorithms multiplied by the redundancy of data structures is a constant value $C \times R = \text{Constant}$ in Figure 13 for solving equivalent problems. Using this rule to synthesize the fault simulation mechanism made it possible to reduce the algorithm's computational complexity to three operations on smart and explicit data structures due to their high level of redundancy. At the same time, the fault simulation process is reduced to zero operations, thanks to the superposition of smart data compiled from two truth tables and one deductive matrix. The simplest implementation of vector-logical fault as address simulation starts with functionality with a single variable as shown in Figure 14. Based on the logical vector, a deductive matrix and a map of tests and repairs are constructed (00, 01, 10, 11).

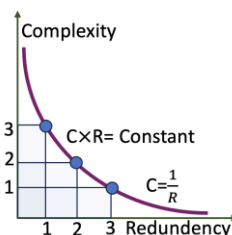


Figure 13. The relationship between complexity and redundancy

	F	D	H	A
0	0 1	0 1		0 0
1	0	1	0 1	0
	1		1 0	0

	F	D	H	A
0	0 1	0 1		0 1
1	0	1	0 1	0
	1	0	1 0	1 1

	F	D	H	A
0	0 1	0 1		1 0
1	0	1	0 1	1
	1	0	1 0	0 1

	F	D	H	A
0	0 1	0 1		1 1
1	0	1	0 1	1
	1	0	1 0	1

Figure 14. Synthesis of a test map for logical functions of a single variable

The following are the results of a Python application designed to verify data structures and fault modeling algorithms. The fault testing map of the most common functions of two variables in Figure 15 given by logical vectors is presented: $0111 - L=x_1 \vee x_2$, $0001 - L=x_1 x_2$, $1011 - L=x_1 \vee \bar{x}_2$, $1101 - \bar{x}_1 \vee x_2$, $0110 - L=\bar{x}_1 x_2 \vee x_1 \bar{x}_2$, $L=1001 - L=\bar{x}_1 \bar{x}_2 \vee x_1 x_2$. The primary purpose of a fault testing map is to simulate (stuck-at and multiple) faults without simulation on input test sets. The H-matrix is the key for fast synthesis of the deductive matrix and fault testing map. The H-matrix is the same for all logical functionalities from n variables. A synthesis of maps for testing the essential logical functions of two variables is presented. This information can be helpful for engineers and students who wish to master the mechanisms of fault simulation, as the truth table addresses. Having learned this lesson, a specialist can manually build a test map for any functionality. Alternatively, a test map synthesis program can be created in a few hours, where the input data is a logical vector, and that is it.

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 1 1 1
00 .1 1. 11	00 1 1 1	0 1 2 3	0 1 1 1
01 .0	01 1	1 0 3 2	1 1
10 .0	10 1	2 3 0 1	1 1
11 .00	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 0 0 1
00 .1 1. 11	00 1	0 1 2 3	0
01 .0	01 1	1 0 3 2	0
10 .1	10 1	2 3 0 1	0
11 .0 0. 00	11 1 1 1	3 2 1 0	1 1 1 1

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 0 1 1
00 .1	00 1	0 1 2 3	1 1
01 .0 1. 10	01 1 1 1	1 0 3 2	0 1 1 1
10 .0	10 1	2 3 0 1	1 1
11 .0	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 1 0 1
00 .1	00 1	0 1 2 3	1 1
01 .0 1. 10	01 1 1 1	1 0 3 2	1 1
10 .1 0. 01	10 1 1 1	2 3 0 1	0 1 1 1
11 .0	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 1 1 0
00 .1 1.	00 1 1	0 1 2 3	0 1 1
01 .0 1.	01 1 1	1 0 3 2	1 1 1
10 .1 0.	10 1 1	2 3 0 1	1 1 1
11 .0 0.	11 1 1	3 2 1 0	0 1 1

Test-map	D-matrix	H-matrix	A-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 0 0 1
00 .1 1.	00 1 1	0 1 2 3	1 1 1
01 .0 1.	01 1 1	1 0 3 2	0 1 1
10 .1 0.	10 1 1	2 3 0 1	0 1 1
11 .0 0.	11 1 1	3 2 1 0	1 1 1

Figure 15. Fault testing map synthesis for the 2-input logic

The second property is that, with a fault testing map, minimal tests can be obtained easily by finding the minimum coverage of all input faults with the minimum number of input test sets. The second applicable property of the fault testing map is that, without synthesis, it defines minimized input test sets that detect the input stuck-at faults of a logic element. Minimum test synthesis is based on covering faults (stuck-at-0, stuck-at-1) of the input lines specified in the coordinates of the matrix with the minimum number of test input sets.

For the Schneider circuit [32] in Figure 16, a deductive matrix in Figure 17 was constructed using the logical vector 1000000000000001. In this example, the advantages of the proposed fault simulation mechanism are emphasized, which consists of the invariance of four simple procedures for building a fault testing map to the dimensions of the logical vector. The only inconvenience for a person is the large size of the fault testing map. Therefore, to prove the validity of the simulation mechanism, we used simple functionalities that emphasize the beauty and simplicity of this technology regarding existing analogs in the world.

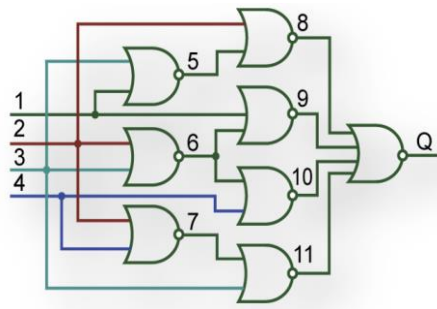


Figure 16. Symmetrical Schneider circuit

D-matrix																	H-matrix																	Logic vector															
TT	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	L	1	0	0	0	0	0	0	0	0	0	1				
0000		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1	1	1	1	1	1	1	1	1	1	1	1	1			
0001			1															1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14	0	1														
0010				1														2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13	0	1														
0011					1													3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12	0	1														
0100						1												4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11	0	1														
0101							1											5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10	0	1														
0110								1										6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9	0	1														
0111									1	1								7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	0	1														
1000										1	1							8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7	0	1														
1001											1							9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6	0	1														
1010												1						10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5	0	1														
1011													1					11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4	0	1														
1100														1				12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3	0	1														
1101															1			13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2	0	1														
1110																1		14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1	0	1														
1111																	1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	1	1														

Figure 17. Deductive matrix synthesis of the Schneider circuit

Based on the deductive matrix, a fault testing map in Figure 18 is built to simulate the faults of input variables without simulation and synthesize the minimum test by finding the optimal fault coverage of the input test sets. As it turns out, the minimum test to detect all the faults of the input functionality variables will be the minimum test to detect all the faults of the circuit's input and output lines. The minimum test input sets are six test vectors: 0000, 0001, 0111, 1000, 1110, 1111.

The fault testing map of the Schneider circuit corresponds between the input test sets and the faults to be detected, which are placed in the cells of the matrix. When the tests are placed in the cells of the matrix, it is not difficult to build an alternate match. Such a matrix would focus more on solving the coverage problem to find the minimum test.

The development of this FSM is supposed to create trivial technological algorithms for processing circuits, including structures with global feedback. The mechanism's advantages are the simplicity and clarity of smart data structures and the ease of human understanding of creating a fault testing map for simulating faults and generating tests.

Smart data structures and fault simulation algorithms are implemented in Python code. The complexity of the program code for implementing the fault simulation mechanism is estimated at 500 lines. The algorithms were verified on several dozen functional elements, for which fault testing maps were built. The FSM takes only 90 minutes to teach students.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TT	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000		...1	..1.	..11	.1..	.1.1	.11.	.111	1...	1..1	1.1.	1.11	11..	11.1	111.	
0001		...0													111.	
0010			..0.											11.1		
0011				..00									11..			
0100					.0..							1.11				
0101						.0.0					1.1.					
0110							.00.			1..1						
0111								.000	1...							
1000								.111	0...							
1001							.11.			0..0						
1010						.1.1					0.0.					
1011					.1..							0.00				
1100				..11									00..			
1101			..1.											00.0		
1110		...1													000.	
1111		...0	..0.	..00	.0..	.0.0	.00.	.000	0...	0..0	0.0.	0.00	00..	00.0	000.	

Figure 18. Fault testing map for input variables of the Schneider circuit

6. CONCLUSION

In-memory computing architecture is proposed based on read-write transactions over smart data structures obtained from logical vectors for fault testing. A deductive synthesis fault testing map for logic is proposed, which, without simulation, allows the determination of all faults detected on test sets and the determining test sets to check specified faults. A superposition of smart data structures is synthesized, containing a deductive matrix D as a derivative of the logical vector L , test truth table T , and fault truth table F . The deductive matrix is seen as the gene of functionality that solves all the problems of testing and verification. In the matrix synthesis, an axiom is used: all the mentioned tables are identical in shape to each other and constantly interact with each other convolutionally $T \oplus L \oplus F = 0$. A universal deductive reversing converter, “test-faults” and “faults-test” for logical functionalities of any dimension, is proposed. Converter functions: fault simulation on test sets $T \rightarrow F$ and synthesis of test sets $F \rightarrow T$ to detect the specified faults. The converter can be used as a test generation and fault simulation service for IP-core SoC under the IEEE 1500 SECT standard. Based on the deductive matrix, a fault testing map for logic is built, where each test set is matched with the detected stuck-at and multiple faults of the input lines. This fault testing map is designed for automatic test or test-bench synthesis, covering all critical points of verifiable functionality defined by 1-unit values in the deductive matrix. Automatic test-bench synthesis can substantially reduce the time-to-market of a digital product with valid Yield parameters.

The metric of the novelty of the proposed study: A new scientific direction of vector logical computing (VLC) is proposed. This is processor-free computation in memory based on read-write transactions on logical vectors. New vector logic computing metrics for fault modeling and simulation are represented: i) professorless data processing means energy-saving computing technology, ii) the absence of the classical von Neumann architecture means no iterations between the data bus between memory and the ALU, significantly reducing data processing time, iii) read-write transactions instead of a robust processor instruction set that allows you to organize the computational process on any memory device, iv) logical vectors assume their simple placement in memory without time-consuming synthesis into a technologically permitted system of elements, v) an address is an attribute of memory and truth tables and a fault as address simulation mechanism, vi) smart data structures reduce the algorithm for their analysis to linear computational complexity, vii) all digital functionality testing tasks are solved using a logical vector and its derivatives, viii) the simulation of tests and faults on the truth table address does not include the simulation algorithm, ix) everything set out in the metric of vector-logical computing, as well as models, methods, algorithms, and mechanisms for fault as address simulation, is new and has no global analogs, and x) the proposed study’s advantage is that it reduces models and algorithms to simple engineering mechanisms.

Future research directions involve fault-in-memory digital circuit simulation with feedback using only read-write transactions. A new faults-as-address simulation (FAAS) Technique will be presented that leverages input faults combination as the address of deductive vector bits, which forms the output vector of the detected input faults in logic elements. The FAAS technique is proposed for digital circuit simulation, where logical vectors represent elements as a compact form of the truth table.




REFERENCES

- [1] Y. Zorian and S. Shoukourian, "Embedded-memory test and repair: infrastructure IP for SoC yield," *IEEE Design & Test of Computers*, vol. 20, no. 3, pp. 58–66, May 2003, doi: 10.1109/MDT.2003.1198687.
- [2] S. Moazzeni, A. Emami, and S. Poormozaffari, "An optimized simulation-based fault injection and test vector generation using VHDL to calculate fault coverage," in *2009 10th International Workshop on Microprocessor Test and Verification*, Dec. 2009, pp. 55–60, doi: 10.1109/MTV.2009.22.
- [3] I. Pomeranz and S. M. Reddy, "Hazard-based detection conditions for improved transition fault coverage of functional test sequences," in *2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 2009, pp. 358–366, doi: 10.1109/DFT.2009.11.
- [4] B. G. Oomman, Wu-Tung Cheng, and J. Waicukauski, "A universal technique for accelerating simulation of scan test patterns," in *Proceedings International Test Conference 1996. Test and Design Validity*, pp. 135–141, doi: 10.1109/TEST.1996.556955.
- [5] I. Pomeranz, S. M. Reddy, and Xijiang Lin, "Experimental results of forward-looking reverse order fault simulation on industrial circuits with scan," in *Proceedings 10th Asian Test Symposium*, p. 467, doi: 10.1109/ATS.2001.990334.
- [6] 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.
- [7] M. H. Assaf, L.-A. Moore, S. R. Das, E. M. Petriu, and S. N. Biswas, "IP core logic fault test simulation environment," in *2010 IEEE Instrumentation & Measurement Technology Conference Proceedings*, 2010, pp. 742–747, doi: 10.1109/IMTC.2010.5488199.
- [8] S. Ozev and A. Orailoglu, "Test selection based on high level fault simulation for mixed-signal systems," in *Proceedings 18th IEEE VLSI Test Symposium*, pp. 149–154, doi: 10.1109/VTEST.2000.843839.
- [9] H. Takahashi, K. O. Boateng, K. K. Saluja, and Y. Takamatsu, "On diagnosing multiple stuck-at faults using multiple and single fault simulation in combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 3, pp. 362–368, Mar. 2002, doi: 10.1109/43.986429.
- [10] P. Bernardi, D. Piumatti, and E. Sanchez, "Facilitating fault-simulation comprehension through a fault-lists analysis tool," in *2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS)*, Feb. 2019, pp. 77–80, doi: 10.1109/LASCAS.2019.8667573.
- [11] T. Hosokawa, H. Takano, H. Yamazaki, and K. Yamazaki, "A diagnostic fault simulation method for a single universal logical fault model," in *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, Jan. 2017, pp. 217–218, doi: 10.1109/PRDC.2017.38.
- [12] I. Pomeranz, "RETRO: reintroducing tests for improved reverse order fault simulation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 8, pp. 1930–1934, Aug. 2020, doi: 10.1109/TVLSI.2020.2997762.
- [13] V. Melikyan, A. Harutyunyan, V. Davtyan, D. Revazyan, and G. Harutyunyan, "Tuning genetic algorithm parameters for placement of integrated circuit cells," in *2023 IEEE East-West Design & Test Symposium (EWDTS)*, Sep. 2023, pp. 1–4, doi: 10.1109/EWDTS59469.2023.10297063.
- [14] P. Wang, A. M. Gharehbaghi, and M. Fujita, "Automatic test pattern generation for double stuck-at faults based on test patterns of single faults," in *20th International Symposium on Quality Electronic Design (ISQED)*, Mar. 2019, pp. 284–290, doi: 10.1109/ISQED.2019.8697831.
- [15] I. Pomeranz, "Test compaction by backward and forward extension of multicycle tests," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 242–246, Jan. 2021, doi: 10.1109/TVLSI.2020.3028005.
- [16] W. Gharibi, A. Hahanova, V. Hahanov, S. Chumachenko, E. Litvinova, and I. Hahanov, "Vector-deductive memory-based transactions for Fault-as-address simulation," *Elektronoe modelirovanie*, vol. 45, no. 1, pp. 3–26, Mar. 2023, doi: 10.15407/emodel.45.01.003.
- [17] W. Gharibi, V. Hahanov, S. Chumachenko, E. Litvinova, I. Hahanov, and I. Hahanova, "Vector-logic computing for faults-as-address deductive simulation," *IAES International Journal of Robotics and Automation (IJRA)*, vol. 12, no. 3, pp. 274–288, Sep. 2023, doi: 10.11591/ijra.v12i3.pp274-288.
- [18] 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.
- [19] 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.
- [20] V. Hahanov, *Cyber physical computing for IoT-driven services*. Cham: Springer International Publishing, 2018.
- [21] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, "8T SRAM Cell as a multibit Dot-product engine for beyond Von Neumann computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2556–2567, Nov. 2019, doi: 10.1109/TVLSI.2019.2929245.
- [22] W. Jiang *et al.*, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 595–605, Apr. 2021, doi: 10.1109/TC.2020.2991575.
- [23] H. Veluri, Y. Li, J. X. Niu, E. Zamburg, and A. V.-Y. Thean, "High-throughput, area-efficient, and variation-tolerant 3-D in-memory compute system for deep convolutional neural networks," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9219–9232, Jun. 2021, doi: 10.1109/JIOT.2021.3058015.
- [24] T. Kanamoto *et al.*, "A single-stage RISC-V processor to mitigate the Von Neumann Bottleneck," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2019, pp. 1085–1088, doi: 10.1109/MWSCAS.2019.8884919.
- [25] M. Yayla, S. Thomann, S. Buschjager, K. Morik, J.-J. Chen, and H. Amrouch, "Reliable binarized neural networks on unreliable beyond Von-Neumann architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2516–2528, Jun. 2022, doi: 10.1109/TCSI.2022.3156165.
- [26] Z. Lin *et al.*, "Two-direction in-memory computing based on 10T SRAM with Horizontal and vertical decoupled read ports," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 9, pp. 2832–2844, Sep. 2021, doi: 10.1109/JSSC.2021.3061260.
- [27] M. Natsui and T. Hanyu, "MTJ-based nonvolatile logic-in-memory circuit with feedback-type equal-resistance sensing mechanism for ternary neural network hardware," in *2019 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Oct. 2019, pp. 1–2, doi: 10.1109/S3S46989.2019.9320674.
- [28] T. Liu, Y. Zhou, Y. Zhou, Z. Chai, and J. Chen, "Hardware efficient reconfigurable logic-in-memory circuit based neural network computing," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2023, pp. 1–5, doi: 10.1109/ISCAS46773.2023.10181471.
- [29] F. Staudigl *et al.*, "X-Fault: impact of faults on binary neural networks in memristor-crossbar arrays with logic-in-memory computation," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 2022, pp.




- 174–177, doi: 10.1109/AICAS54282.2022.9869897.
- [30] N. K. Macha, P. Samant, and M. Rahman, “Crosstalk logic circuits with built-in memory,” in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2021, pp. 79–83, doi: 10.1109/ISVLSI51109.2021.00025.
 - [31] T. Zanolli, F. M. Puglisi, and P. Pavan, “Reconfigurable smart in-memory computing platform supporting logic and binarized neural networks for low-power edge devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 478–487, Dec. 2020, doi: 10.1109/JETCAS.2020.3030542.
 - [32] 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.
 - [33] E. G. Ulrich and T. Baker, “The concurrent simulation of nearly identical digital networks,” *Papers on Twenty-five years of electronic design automation*, pp. 318–323, 1988.
 - [34] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital systems testing and testable design*. New York: Computer science press, 1990.
 - [35] Riahi, Navabi, and Lombardi, “A VPI-based combinational IP core module-based mixed level serial fault simulation and test generation methodology,” in *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat No 03CH37417) ATS-03*, 2003, pp. 274–277, doi: 10.1109/ATS.2003.1250822.

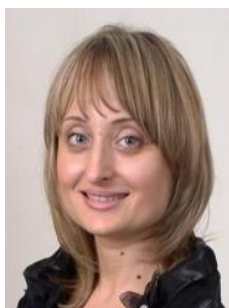
BIOGRAPHIES OF AUTHORS






Vladimir Hahanov    was born in USSR in 1953. He is a Doctor of Science, Professor of Computer Engineering Faculty, Design Automation Department, Kharkov 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 36 PhDs. 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 198 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, IFAC member, and Communication Society member. He can be contacted at hahanov@icloud.com.






Wajeb Gharibi    is a professor of Computer Science Electrical Engineering, University of Missouri Kansas City, USA. His research and development fields include design and test 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.



Svetlana Chumachenko    was born in USSR in 1969. She is a Doctor of Technical Sciences, Professor, and Head of the Design Automation Department, Kharkov 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 2012–2016. 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, Kharkov 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 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.