

Faults-as-address simulation

Vladimir Hahanov, Svetlana Chumachenko, Eugenia Litvinova, Ivan Hahanov,
Veronika Ponomarova, Hanna Khakhanova, Georgiy Kulak

Design Automation Department, Kharkov National University of Radio Electronics, Kharkov, Ukraine

Article Info

Article history:

Received Apr 5, 2024

Revised Aug 13, 2024

Accepted Aug 27, 2024

Keywords:

Fault simulation matrix

Fault truth table

Fault-as-address simulation

In-memory computing

Logic vector computing

Smart data structures

Test truth table

ABSTRACT

Fault-as-address-simulation (FAAS) is a simulation mechanism for testing combinations of circuit line faults, represented by the bit addresses of element logical vectors. The XOR relationship between the test set T and the truth table L of the element forms a deductive vector for fault simulation, using truth table addresses or the logic vector bits. Addresses are used in the simulation matrix to mark those n-combinations of input faults detected at the element's output. The columns of the simulation matrix are treated as n-row addresses to generate an element output row via a deductive vector. There is no transport of input faults to the element output, Only the 1-signals written in the non-input row coordinates of the circuit simulation matrix. The simulation matrix is initially filled with 1-signals along the main diagonal. The line faults detected on the test set of circuits are determined by the inverse of lines good values, which have 1-values in the matrix row corresponding to the output circuit element. The deductive vector is obtained by the XOR-relations between the test set and logical vector in three table operations. The advantage of the proposed FAAS mechanism is the predictable complexity of the algorithm and memory consumption for storing data structures when simulating a test set.

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



Corresponding Author:

Vladimir Hahanov

Design Automation Department, Kharkov National University of Radio Electronics

Kharkov, Ukraine

Email: hahanov@icloud.com

1. INTRODUCTION

Statement of the problem the paper is addressing. Most of the existing solutions for processing big data, including design and verification, use potent computers, networks, and computing centers where arrays of information are stored. Such solutions require a lot of electrical energy and are close to power plants. Given that there will always be insufficient electricity, everything related to the energy-intensive processing of big data will gradually become a thing of the past. Instead, there is a steady increase in in-memory computing, which will solve the problems of intelligent processing of big data on smart data structures without programming algorithms and architectures free of potent processors. One of these cost-effective solutions is proposed for research: in-memory modeling and simulation on smart data structures that use only read-write transactions. The study's formula is the use of smart data structures in vector-table forms of logic circuit description to organize in-memory computing based on read-write transactions [1] for simulating faults as addresses [2], [3]. Experts in computer technology, including Gartner, predict a shift to mass computing in memory and the abandonment of the von Neumann architecture [2] and powerful processors [4], [5]. They advocate the gradual transition of computing to any substance that can store data. To perform any computational actions, two read-write transactions on memory elements are enough [1], [2]. This is the practical essence of the proposed study. On the other hand, intelligent computing, according to leading

experts, has a strong tendency to use smart memory-based data structures [4], [6] to process big data without programming algorithms [7], [8]. At the same time, smart data structures can be created once by a superposition of explicit computing components or by training a machine learning (ML) model when the data structure acquires valuable properties for a successful solution to the problem. In both cases, the computational complexity of the smart data structure processing algorithm tends to zero due to the significant redundancy of data structures. The economy of in-memory mass computing involves a substantial reduction in energy (78%) [2] for processing big data and the time (32%) [4] to solve problems due to the speed of read-write transactions at the level of one nanosecond [5].

Here are what has been done before about the problem. Models and algorithms are strongly interconnected. Redundancy of one component leads to a decrease in the other and vice versa. If an engineer wants a simple algorithm, he should use the redundancy and simplicity of explicit data structures (logical vectors, truth tables [1], [3], [9], and matrices) instead of analytical [10]–[12], or graph [13]–[15] models. Design and testing have been, are, and will continue to be the most advanced technology for building intelligent computing [10], [16], the goal of which is to create the cyber brain of humanity. An urgent problem is verifying new digital solutions using system-on-chip (SoC) numbering billions of transistors. Automatic synthesis systems of Synopsys, cadence can design a system of any complexity. However, the question is how to verify such a system. How can a test for it be built to determine its qualities in detecting logical or physical faults? Electronic design automation (EDA) applies testable design standards to break down a complex system into the market's SoC intellectual property (IP) cores. Tests are then built for these modules, which must be verified using fault simulation algorithms. Typically, these algorithms use compact analytical or hardware description language (HDL) digital circuits and element models. Their processing involves the creation of powerful compilers based on the use of processor instructions. Such simulators are costly and require a lot of time and energy [17], [18]. What can be offered in return? Technologically simple, cheap-to-use test generators and simulators [19]–[21] that consume a minimum amount of energy [22]–[24]. The research formula for this metric is to develop mechanisms for in-memory modeling and simulation of faults-as-addresses based on logical vectors [21], [25] of circuit elements, which are processed based on read-write transactions [20], [26]. For this purpose, redundant smart and explicit data structures [20], [27], [28] have been introduced and used, significantly reducing the computational complexity of simulation algorithms [1], [15], [29]. Logical vectors, truth tables, and logical matrices represent smart data structures. Only one XOR operation and a read-write transaction are required to process them for fault-as-address simulation. However, this XOR operation can also be reduced to read-write transactions on a logical vector [1], [19]. The efficiency of the simulation mechanism is unparalleled, providing simplicity and good economics of the solution. Computer engineering students master this mechanism in one lesson, 45 minutes. The topic of fault simulation in the EDA market has been relevant for 70 years and occupies one of the first places in the development of researchers and companies. The market relevance of this topic is determined by the following metrics [29]–[31]: i) the development of smart data structures and efficient algorithms for testing increasingly complex computer systems and networks consisting of billions of equivalent gates; ii) invariance of structures and algorithms concerning technologies and types of digital products that change rapidly over time [32]; iii) the flexibility of data structures and algorithms will allow the processing of a wide range of digitized technical devices and processes for their testing, verification, diagnostics, and fault detection [33]–[35]; iv) considering faults as big data, algorithms for their analysis must be efficient in terms of time and energy consumption. Therefore, fault simulation should be implemented on primitive read-write transactions as in-memory computing without the use of the robust instruction set of the universal processor in the von Neumann architecture [2], [36], [37]; v) data structures for fault simulation should be simple and accessible to computers and humans [36]. Such ideal data structures, invariant in time, are the unjustly forgotten truth table and its compact derivative, the logical vector. They are ideal for efficiently solving in-memory computing tasks, including big data processing [38] and fault simulation; vi) The truth table should be elevated to the rank of an ideal internal model of a computational process or circuit that is invariant concerning dozens of languages describing hardware or software [37], [38]; vii) The truth table generates a complete set of logic-fault combinations for arbitrarily complex logical functionality [1], [10], [16]. Thus, combining in-memory computing and smart data structures based on the truth table may be a rational way to solve design and test problems over a long time in the development of global computing [39]–[41].

The proposed solution and the results achieved. The goal is to significantly reduce the latency and energy consumption when simulating faults-as-addresses using in-memory computing technology in logical circuits of any dimension based on read-write transactions over smart data structures built based on a logical vector. The objectives of this research are to i) identify the components of smart compact data structures for logic circuit fault simulation; ii) create a simulation algorithm for faults F of a logical circuit on an input test set T by synthesizing deductive vectors of elements from their logical vectors L , using the equation $D=T\oplus L$; and iii) verify the data structures and algorithms for modeling and simulation faults-as-addresses, using examples of logic circuits. Without going into details, von Neumann's architecture [2] controls and executes

the computational process. Therefore, it can be transferred to memory, where big data resides. Moreover, this architecture can be used to simulate both logic and circuits in two modes: good simulation of test-as-addresses and simulation of fault-as-addresses. The device of control and execution in computing has not yet been canceled [2], [14], [19].

2. FAULTS-AS-ADDRESS CIRCUIT SIMULATION: UNUSUAL AND SIMPLE ENGINEERING MECHANISM

What is new? Faults-as-addresses simulation (FAAS) is a technique that leverages input faults combination as deductive vector bits addresses that form detected input faults in logic element output. FAAS technique is proposed for digital circuit simulation, where logical vectors represent elements as a compact form of the truth table. The truth table is proposed as a smart data structure for fault simulation of input and internal lines of logic circuits. The logical vector in the truth table and the input test set are used to generate a deductive vector that detects any combination of faults in the circuit lines to its external outputs. The novelty of the FAAS mechanism is the Fault Simulation Matrix address filling using deductive vectors, which simulate a combination of faults as an address. The in-memory simulation uses read-write transactions, which makes the FAAS technique a free central processing unit (CPU) instruction system and cost-effective in terms of power and time to simulate faults. The circuit fault simulation contains novelty points: circuit 1 is the deductive vector synthesis for input set and logic vector to detect input faults on element output; circuit 2 is the development of a quadratic fault simulation matrix of the circuit for every test set; circuit 3 is the simulation of circuit lines faults-as-addresses using a simulation matrix and deductive vectors of logic elements to obtain a vector of detected faults on the input test set; and circuit 4 is the formation of a table of detected faults to determine the quality of the input sets and the test. Algorithms and data structures are proposed for in-memory simulating fault-as-addresses and logical schemes of any structural complexity. Smart data structures have three macro components: circuit description, simulation matrix, and fault detection table. The truth table is the longest-lived model of the computational process; it is more than 100 years old, and today, it is practically not used for organizing computations. Underneath the simplicity of the form of the truth table, which is understandable to humans and machines, there is an undeciphered genome of emerging computing. Fault simulation algorithms use read-write transactions on smart data structures in any SoC, field programmable gate arrays (FPGA), application-specific integrated circuit (ASIC), or reduced instruction set computer (RISC-V) memory and do not require processor instructions and pre-synthesis of the circuit to bring logical functionalities to a specific element basis. The market goal of the proposed technique is to solve IP-core verification problems based on in-memory computing using the IEEE 1500 standard for embedded core test (SECT) standard [21]. A data structure metric is a logical vector or truth table. The practical novelty of the FAAS mechanism is the fault simulation matrix address filling using deductive vectors, which simulate a combination of faults as an address as shown in Figure 1. It provides a model of a good-value line behavior circuit using logic vectors and a circuit model for fault simulation using deductive vectors. A quadratic fault simulation matrix is presented, which is filled along the main diagonal with single values identifying faults of the circuit lines. Three truth tables for the synthesis of the deductive vector are also presented. All the listed data structures are required to model the fault as addresses on a single input test set 0011. The result of a good simulation is shown in the G-line of the simulation matrix. The result of the fault simulation is shown in the F-line of the simulation matrix.

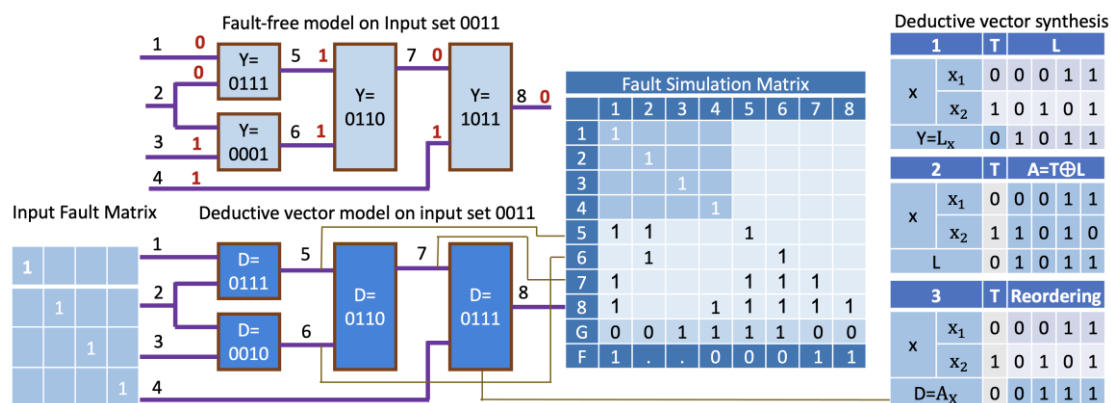


Figure 1. Faults-as-address circuit simulation

The relationship $T \oplus L = D$ between the test set T and the truth table L of the element forms a deductive vector for fault simulation, using addresses the truth table or the logic vector bits. On the other hand, addresses are used in the simulation matrix to mark those n -combinations of input faults detected at the element's output. The columns of the simulation matrix are treated as n -row addresses to generate an output row of the element via the deductive vector. There is no transport of input faults to the output of the element, Only the 1-signals written in the non-input row coordinates of the circuit simulation matrix. The simulation matrix is initially filled with 1-signals along the main diagonal. The line faults detected on the test set of circuits are determined by the inverse of the good values of lines G , which have 1-values in the matrix row corresponding to the output of the last circuit element. The deductive vector is obtained by the XOR-relations between the test set and the logical vector in three table operations. The advantage of the proposed FAAS mechanism is the predictable complexity of the algorithm and the memory consumption for storing data structures when simulating a test set, determined by the formula $Q = N^2 + \sum_{i=1}^n (Y_i + D_i)$, where N , n are number of lines and elements in the circuit and Y , D are a set of logical and deductive vectors.

Construction of the recoding matrix [1] by taking a Cartesian \oplus -square on the addresses of the truth table from n -variables according to the formula: $H = A \oplus A = A_{\oplus}^2$. Addresses act as test sets T and combinations of logic faults F . The resulting matrix is a constant for all logical functions from n -variables. There are several modifications to the recoding matrix construction. Next, a simple recursive mechanism for building a recoding matrix based on prediction is proposed. Four sequential operations are performed on the four quadrants of the matrix: $H_{i+1}^1 = H_i$; $H_{i+1}^2 = 2^n + H_i$; $H_{i+1}^3 = H_{i+1}^2$; $H_{i+1}^4 = H_{i+1}^1$. The construction of a matrix of any dimension starts from 0: $H_0^1 = 0$. Another mechanism for recursively constructing a recoding matrix uses computational history: $H_i^1 = H_{i-1}^1$; $H_i^2 = 2^n - 1 - H_i^1$; $H_i^3 = H_i^2$; $H_i^4 = H_i^1$. The computational complexity of both mechanisms for constructing a recoding matrix for a logical function from n -variables is equal to 2^{n+1} . The data structures reflecting the recursive generation of the matrix are shown in Figure 2.

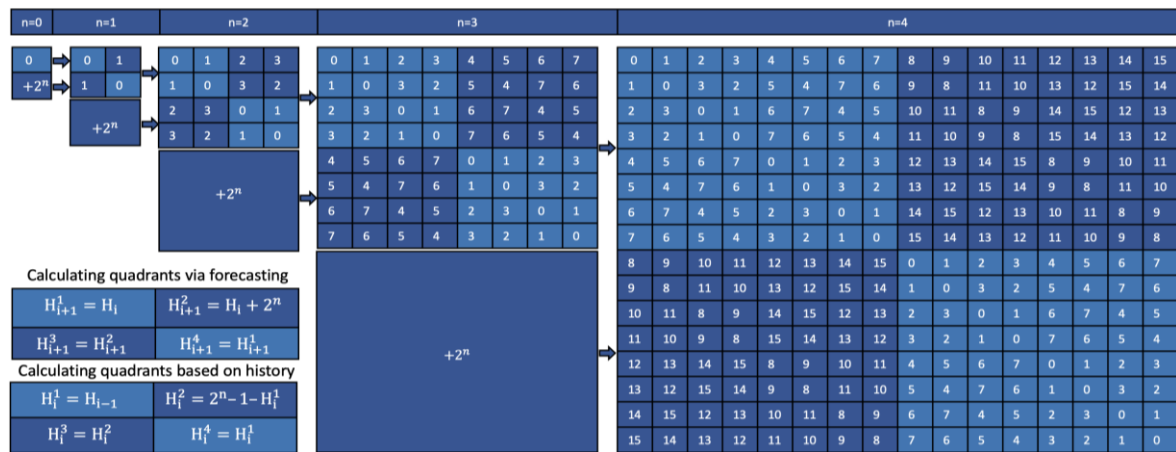


Figure 2. Synthesis of H-matrix recoding

The practical use of this H-matrix is the rapid construction of a deductive vector by recoding the coordinates of the logic vector on the decimal codes of a string with a number determined by the test input set. An example of a deductive vector construction uses logic vector 0111, $n=2$, test set 10. Consider the matrix of two variables on a string number 2 ($T=10$), which is equal to 2301. Recoding the coordinates of a logical vector gives a deductive vector: $D = L_{H-T=2} = 0111_{2301} = 1011$. Using an H-matrix recoding to synthesize deductive vectors speeds up logic circuit fault simulations by 30%. One recoding matrix for an element with the maximum number of variables is needed to synthesize deductive vectors of a logical circuit.

3. DATA STRUCTURES FOR LOGIC ELEMENT AND CIRCUIT FAULT SIMULATION

Vector-logical design and test computing is a processor-free in-memory SoC IP-core testing mechanism based on read-write transactions on logical vectors and their derivatives. A mechanism is a communicating relationship between the redundancy of data structures and the computational complexity of algorithms for their processing when an increase in one of them leads to a decrease in the other and vice versa. Mechanisms for modeling, simulation, and testing digital projects are proposed. The object of the study is to design and test computing based on vector-logic mechanisms located in memory to save energy

and design time. The subject of the research is vector-logical in-memory computing for solving the problems of modeling, testing, and verification of digital projects based on vector-logical models of the IP-core SoC in Figure 3. Here, all computing components are completely new and focus on the EDA market of cost-effective engineering solutions. There's no potent CPU here, not even RISC-V. Only read-write transactions and one vector XOR operation are easily converted to transactions.

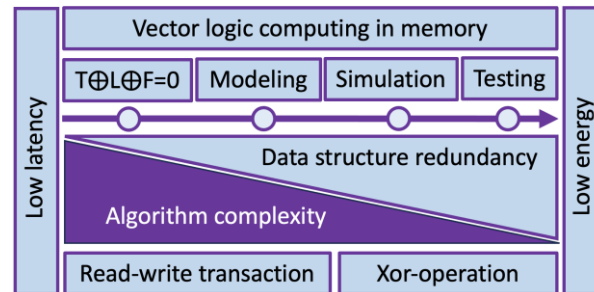


Figure 3. Vector logic design and test computing in memory

The vector-logic computing equation $T \oplus L \oplus F = 0$ is used to solve all testing problems. It defines the relationships between the same component formats (test, logic, faults) to determine any of them by the other two. Such formats can be logical vectors, truth tables, and logical matrices. All these components form smart data structures interconnected to get a result without an algorithm. The problem of modeling data structures is solved by superpositioning those explicit components that provide solutions without an algorithm. This trend is expected in the IT market, which aims to create and train a structure of smart data for intelligent solutions to all problems in cyber-physical space without programming algorithms. All verification technologies use the convolution equation of the relations between $L \oplus T \oplus F = 0$ specification L , implementation T , and errors F (faults). Verification is the process of determining the correspondence between the specification and the synthesized device. This correspondence is then used to eliminate design errors $T \oplus L = F$ by introducing logical-time redundancy. Test verification involves identifying faults $F = T \oplus L$ and designing based on a synthesized test (project redundancy) for a given class of faults. Suitable design standards, such as IEEE 11.49, 1500, and 1687, allow you to reduce the length of the test of IP components of a project and improve its quality by using boundary scan registers [21]. Formal verification is the process of achieving the quality of a project $T \oplus L = F$ based on logical-temporal redundancy (temporal logic) and a library of valid circuit solutions. Disassembling a model is the process of obtaining a digital device model based on the XOR interaction of a test and a given class of faults $L = T \oplus F$. Fault diagnosis is the process of determining the cause, location, and type of defects in a digital circuit. Next, all the components of smart data structures are defined to solve the problems of testing using the characteristic equation and based on the XOR operation.

Components of smart data structures are introduced to develop simulation algorithms. These structures' metrics use the dimensionality of a logical vector determined by the number of input variables. A logical element is any binary functionality of n variables, with an indeterminate internal structure, given by a logical vector, dimension 2^n . Logical vector L is an explicit form of defining functionality using an ordered sequence of 2^n bits, where each bit has its binary address in the metric n variables. A logical vector and an ordered set of explicitly specified binary addresses form a truth table. A logical vector has all the properties of a truth table, while it is more compact and technologically advanced for solving design and test in-memory problems. Input binary test set x is an input test sequence of n binary bits, considered the address of a deductive vector that provides transportation of a combination of an element's input faults to its output. The output state of element Y is a binary value that results from feeding a test binary set x to an element specified by the logical vector L . Test vector refers to concatenation of binary input set x and output state Y , using the functions $T = (xY)$. Active truth table $A = T \oplus L$ is an unordered form of specifying all combinations of input faults by a set of 1-coordinates predetermined by n inverse values of the bits of the input binary set. Fault truth table F is an explicit form of specifying all combinations of input faults by a set of unit coordinates that are additionally determined by n inverse values of the bits of the input binary set. The truth table of the T -tests is an explicit form of defining all 2^n combinations of input binary test cases for a logical element. A functional coverage truth table is an explicit form of specifying all combinations of input binary test sets and the 2^n state of logical element outputs. A functional coverage is essentially a logical vector of an element. Deductive D -vector, dimension 2^n , points with its single bits to the columns with the faults to be detected at

the 1-coordinates of the fault truth table. Deductive matrix, dimensionality $2^n \times 2^n$ is a gene of logical functionality that contains complete information about all faults of a functional element detected on a test set. Smart data structure (for fault simulation) is the explicit data structures (logical vectors, truth tables, deductive matrices) interconnected by the unified metric of the logical vector space 2^n . Smart data structure does not require conditional processing. The truth table, as a smart data structure, helps to exclude conditional IF statements from the program code. Since the truth table is the complete set of conditional if statements (in each variable metric) translated into the table of ordered addresses of the bits of the logical vector. Circuit description (circuit input set data) contains information about the relationships between the schematic elements (the number of inputs of each element and their numerical identifiers) and logical and deductive vectors for performing the circuit fault simulation. Simulation matrix (matrix of circuit fault simulation on input set) refers to quadratic matrix of the schematic structure for stuck-at-faults simulation of input, internal, and output lines, dimension, of the number of lines of the circuit, determined by 1-values of the coordinates of the main diagonal. It is designed to model circuit faults as addresses by using deductive vectors of logic elements to obtain a vector of faults to be detected on the input set. End-to-end numbering of schematic lines makes tracing structural features (reconvergence fanouts) easy in fault analysis based on a simulation N-squared matrix. The table for test-detected circuit faults contains integral information about the faults detected on each input set and an assessment of the quality of the input set and the test. It is used to validate the test and find faults in the digital device. The formula for simulating stuck-at-faults of circuit lines is as follows: from the input set on the external inputs and numbered N-lines of the circuit is synthesized N^2 simulation matrix filled with 1-coordinates diagonally to obtain a vector of detected faults on the N-lines of the circuit by constructing a matrix of deductive vectors from the m-number of functional elements of the circuit, based on logical vectors and binary sets at the inputs of the elements, after which each output of the schematic element is simulated by the sequential processing of the combination of lines of the simulation matrix at the element inputs as addresses of the cells corresponding to the corresponding deductive vectors, after which a disjunction operation is performed on those lines of the simulation matrix that belong to the external interface outputs of the circuit, in order to obtain an integral V-vector of the simulation, the 1-coordinates of which are further determined by the inverse values of the lines of the vector of the good simulation to obtain the vector of tested faults on the input binary set. Faults-as-addresses simulation is a technique that leverages input fault combinations as the addresses of deductive vector bits, which forms the output vector of the detected input faults in a logic element. The model of detected line fault is represented by a 1-signal in the simulation matrix, and the fault that was not detected is noted by a 0-signal.

4. INPUT DATA MODELS FOR LOGICAL ANALYSIS

Input data models for logical processing can be represented in Figure 4. Number 1 is the set of compact unstructured data requiring complex and sequential input processing [42]. Number 2 is the binary vector of unitary-encoded data that uses complex processor register variables to process pairs of input vectors in parallel. Number 3 is the addresses formed by the columns of the input matrix of unitary-encoded data to perform read-write in-memory transactions on the bits of the logical vector (without traditional processor logic) with parallel analysis on the columns of data as addresses. Here, the vector $L = 01101001$ is an XOR function of three variables that form addresses from the data to be analyzed. The longer the logical vector length, the greater the degree of parallelism when processing big data as addresses. Logical vectors do not need to be reduced to the logic of the processor using rather complex synthesis. In-memory computing based on logic vectors is free from synthesis and traditional CPU logic, replaced by read-write transactions (600 ps latency), which are not inferior in speed to the arithmetic logic unit (ALU) [38]. At the same time, in-memory computing has half the power consumption when processing big data compared to a processor [39].

Two data processing options represent fault-as-address simulation techniques as shown in Figure 5.

- Technique 1. Use the columns of the S-matrix simulation along the input lines of the element as the addresses of the deductive vector bits, which form the coordinates of the vector of the faults to be detected on the output line $Y_{i,j} = D_{x_j}$. For this example, it would be $Y_{12} = 100001000100$. Here, each 1-unit is defined by address 100 in the truth table of the deductive vector, where $D(100)=1$. This technique is implemented in the FAAS engine for circuit fault simulation.
- Technique 2. Distribute all columns of the S-matrix of the simulation with the input numbers of the circuit lines to the addresses of the deductive vector truth table, and then generate the 1-coordinates of the fault vector to be tested on the output line using the numbers of the lines equivalent to the columns, if the column of the truth table is covered by the 1-coordinate of the deductive vector. Both techniques for fault simulation of circuit lines as addresses have the same computational complexity $Q=n \times N$ (where n is the number of inputs of the element being processed and N is the number of lines of the circuit). The second technique has a valuable property: the faults equivalence of circuit lines by the

columns of the truth table, which is used to diagnose defects in the circuit. Another essential feature of the proposed techniques is that in the simulation matrix, the tested faults of both signs (stuck-at-1, stuck-at-0) are represented by a 1-coordinate, which is very convenient for generating addresses for their parallel processing on deductive vectors. 1-coordinates are further determined by specific faults, inverse to the good state of the lines, after the formation of the entire matrix of the circuit simulation. The first technique for fault simulation of elements as part of a circuit is focused on implementation in the EDA market using in-memory computing technology. This technique can also be effectively used for parallel processing of big data, such as addresses, to solve problems of determining similarity–difference and equivalence of patterns.

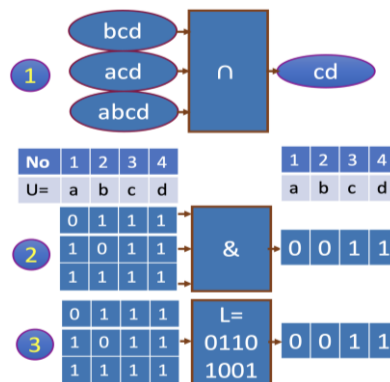


Figure 4. Input data models for logical analysis

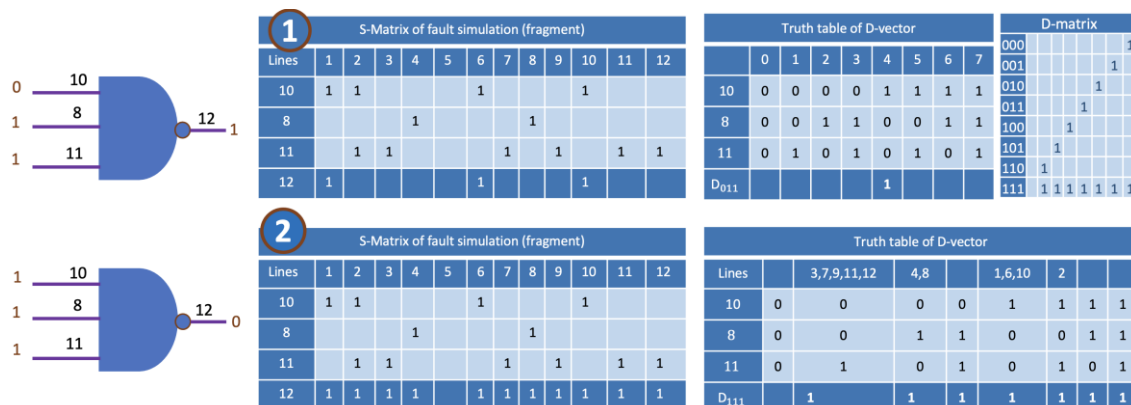


Figure 5. Techniques for fault simulation as addresses on a deductive vector

The fault simulation matrix of circuit lines as addresses is the basis of the simulation engine in Figure 6. The circuit fault simulation matrix on the input test set has dimensions $N \times N = N^2$, where N is the number of lines in the circuit. It is filled with one unit along the main diagonal, which, in the simulation process, turns into binary vectors of the faults to be detected for each output number of the element.

The 1-coordinate of the matrix encodes the fault detection on the circuit line of both fault types (stuck-at-0, stuck-at-1). Vector Sum V performs the function of logically concatenating those rows of the simulation table that are the external outputs of the circuit. In this example, there is one external circuit input, number 12. The Good vector shows the state of all circuit lines in the absence of faults. The faults vector identifies faults (stuck-at-0, stuck-at-1) that are detected on a test set using a simple rule: If $(\text{Sum}V)_i = 1$, then $(\text{Faults})_i = \overline{\text{Good}}_i$. This expression can be implemented using the truth table shown at the bottom of Figure 6. The fault vectors form the result of the simulation, which is the table of test-detected circuit faults. To form the address of a deductive vector bit, the coordinates of those strings whose numbers form the element's inputs are used to form the address of the bit. Each deductive vector of element forms its output row of the simulation matrix using eventfulness rules, a set of empty cells or null addresses are not processed. To obtain

vector line 12 for the element shown in Figure 6, you need to concatenate the addresses by input numbers term (10,8,11): 101, 111, 011, 000, ... The computational complexity of processing the simulation matrix on the test set is determined by the complexity of address concatenation. It is equal to $\frac{1}{2}N^2$, where N is the number of lines in the circuit.

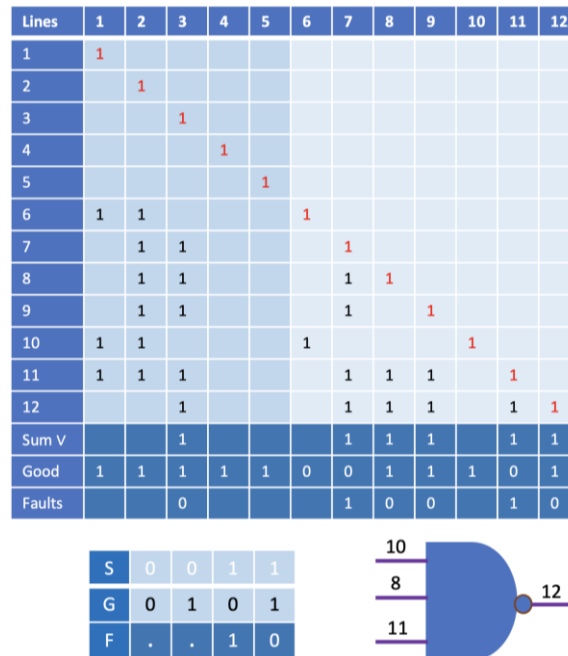


Figure 6. Matrix of fault simulation of circuit lines as addresses on the input set

5. FAULTS AS ADDRESS SIMULATION IN THE LOGIC CIRCUIT

The problem of transferring the von Neumann architecture to memory and replacing the powerful processor on read-write with transactions on logical and deductive vectors to reduce energy and time consumption when logic circuit simulation of any dimension is solved. The goal is to develop a FAAS mechanism for fault-as-address simulation for digital circuits of any dimension. The tasks as presented in Figure 7 are as follows. Task 1 is synthesis of deductive vectors for transporting input faults to the output of an element on an input test set. Task 2 is development of a quadratic matrix of the circuit structure to simulate stuck-at-faults of input, internal, and output lines. Task 3 is simulation of circuit faults-as-addresses using simulation matrix and deductive vectors of the logic element to obtain a vector of the detected faults. The end-to-end numbering of circuit lines makes tracing structural features (reconvergence fanouts) easy in fault analysis using a simulation matrix. Task 4 is formation of a table of detected faults to determine the quality of the input set and the test.

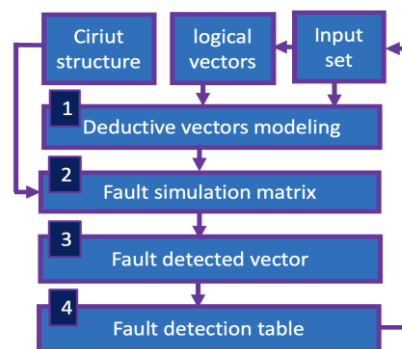


Figure 7. Circuit fault simulation structure

Truth table synthesis of the H-matrix line for deductive vector creation in Figure 8. The coordinates of each row of the H-matrix [10] of bit recoding are represented by decimal address codes, which are obtained by XOR interaction of the input binary test set T and the columns of the truth table $L: A = T \oplus L$ of any functionality from the n -variables. This is the theoretical essence of the synthesis of the bit-recoding matrix. The results of the XOR interaction of the input binary test vectors ($T=0000, 0111$) and $2^n=16$ columns of the truth table form the 0 and 7 rows of the bit-recoding matrix to generate a deductive vector. Each table shows the result of the XOR summation of the test set T and the standard truth table L . The upper part of each table is represented by the recoding vector obtained to compile the H-matrix. The binary-decimal code of the test set forms the corresponding string of the recoding matrix. Knowing the recoding vector H_{0111} it is easy to obtain the deductive vector D by executing the formula: from the test set $T=0111$ as the bit address of the logical vector $L_{0111} = 1$, the state of the element's output is determined $Y=1$, XOR-summation of which with all the bits of the logical vector $L=1110001111000111$ gives a vector of activity $A=Y \oplus L=0001110000111000$, to which the H-vector of the recoding of the bits on the test set $T=0111$ is applied to obtain the deductive vector $D=A_H=0011100000001110$ whose bits are already written to the binary-decimal addresses standard. The H-matrix of bit recoding is the key to constructing a matrix of deductive vectors of any logic.

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 | 9 | 8 | 11 | 10 | 13 | 12 | 15 | 14 |
| 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 | 10 | 11 | 8 | 9 | 14 | 15 | 12 | 13 |
| 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 11 | 10 | 9 | 8 | 15 | 14 | 13 | 12 |
| 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 12 | 13 | 14 | 15 | 8 | 9 | 10 | 11 |
| 5 | 4 | 7 | 6 | 1 | 0 | 3 | 2 | 13 | 12 | 15 | 14 | 9 | 8 | 11 | 10 |
| 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 | 14 | 15 | 12 | 13 | 10 | 11 | 8 | 9 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 9 | 8 | 11 | 10 | 13 | 12 | 15 | 14 | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| 10 | 11 | 8 | 9 | 14 | 15 | 12 | 13 | 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 |
| 11 | 10 | 9 | 8 | 15 | 14 | 13 | 12 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 |
| 12 | 13 | 14 | 15 | 8 | 9 | 10 | 11 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| 13 | 12 | 15 | 14 | 9 | 8 | 11 | 10 | 5 | 4 | 7 | 6 | 1 | 0 | 3 | 2 |
| 14 | 15 | 12 | 13 | 10 | 11 | 8 | 9 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| | | | | | | | | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| H ₀₀₀₀ | T | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| x ₁ | 0 | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| x ₂ | 0 | | | | | | 1 | 1 | 1 | 1 | | | | 1 | 1 | 1 | 1 |
| x ₃ | 0 | | | | 1 | 1 | | | 1 | 1 | | | 1 | 1 | | | 1 |
| x ₄ | 0 | | 1 | | | 1 | | | 1 | | 1 | | | 1 | | | 1 |

| | | | | | | | | | | | | | | | | | |
|-------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|---|---|
| H ₀₁₁₁ | T | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| x ₁ | 0 | | | | | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| x ₂ | 1 | 1 | 1 | 1 | 1 | | | | | 1 | 1 | 1 | 1 | | | | |
| x ₃ | 1 | 1 | 1 | | | 1 | 1 | | | 1 | 1 | | | | 1 | 1 | |
| x ₄ | 1 | 1 | | 1 | | 1 | | 1 | | 1 | | 1 | | | 1 | | 1 |

| | | | | | | | | | | | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|---|
| H ₀₁₁₁ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
| L ₀₁₁₁ = 1 | 1 | 1 | 1 | | | | | 1 | 1 | 1 | 1 | | | | 1 | 1 | 1 |
| A = Y⊕L | | | | | 1 | 1 | 1 | | | | | 1 | 1 | 1 | | | |
| D=A _H | | | | 1 | 1 | 1 | | | | | | | 1 | 1 | 1 | | |
| Standard BDA | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |

Figure 8. Truth table synthesis of the H-matrix line for deductive vector creation

It is proposed to use a truth table to synthesize the deductive vector of a functional element in Figure 9 based on a logical vector and a given binary input set for its use in simulating stuck-at-faults in a digital circuit. The formula for deductive vector modeling is as follows. The input binary set, as the cell address of the logical vector, contains the output state of the element, the concatenation of which with the

input binary forms an input-output state vector for XOR interaction with all the columns of the truth table to synthesize the activity table, to which the procedure of ordering the columns in ascending order of the binary addresses of the input variables is applied to obtain a deductive vector in the ordered activity table.

| | | | | | | | |
|---|----------------|---|-------|---|---|---|--------------------------|
| 1 | | T | F | | | | Truth table for L |
| x | x ₁ | 1 | 0 | 0 | 1 | 1 | |
| | x ₂ | 0 | 0 | 1 | 0 | 1 | |
| Y=L _x | | 1 | 1 | 1 | 1 | 0 | Logical vector L |
| 2 | | T | A=T⊕F | | | | A-table as addresses |
| x | x ₁ | 1 | 1 | 1 | 0 | 0 | |
| | x ₂ | 0 | 0 | 1 | 0 | 1 | Activity Vector A |
| L | | 1 | 0 | 0 | 0 | 1 | |
| 3 | | T | | | | | Standard BD addresses |
| x | x ₁ | 1 | 0 | 0 | 1 | 1 | |
| | x ₂ | 0 | 0 | 1 | 0 | 1 | Deductive Vector D |
| D _i = A _{x_i} | | 1 | 0 | 1 | 0 | 0 | |

Figure 9. Generation of the deductive vector for an element via a truth table

Data structures for logic circuit fault simulation are represented by the following macro-components: circuit description and simulation matrix. The fault detection table consists of i) identifiers of the input-output of the circuit and logic elements, which together make up the interface of the circuit; ii) a matrix of logical vectors that define the functionality of each element of the circuit; iii) a matrix of deductive vectors that form lists of input faults to be detected at the output of each element of the circuit; iv) vector of binary good behavior of all circuit lines in the format of input, internal and output lines; v) quadratic matrix (N^2 of the number of circuit lines) for simulation of circuit faults on an input binary test set, pre-populated with 1-values at diagonal coordinates, creating a stuck-at-fault circuit model; and vi) vector of faults detected on the binary test set, inverse concerning the state of the circuit lines of good behavior. Next, a circuit as in Figure 10 is proposed, for which data structures will be built to execute the fault simulation algorithm. Here is the circuit fault simulation formula: Simulation N^2 matrix filled with unit coordinates diagonally is synthesized from the binary test set on the external inputs and numbered N-lines of the circuit to obtain a vector of detected faults on the N-lines of the circuit by constructing a matrix of deductive vectors based on the m-number of functional elements of the circuit, based on logical vectors and binary sets on the inputs of the elements, after which each output of the circuit element is simulated by sequential processing combination of the lines of the simulation matrix at the inputs of the element as addresses of the cells of the corresponding deductive vectors, after which the disjunction operation is performed on those lines of the simulation matrix that belong to the external interface outputs of the circuit, in order to obtain an integral V –vector of the simulation, the 1-coordinates of which are supplemented by the inverse values of the lines of the vector of the good simulation to obtain the vector of the detected faults on the input binary set. These can be seen in Figure 11.

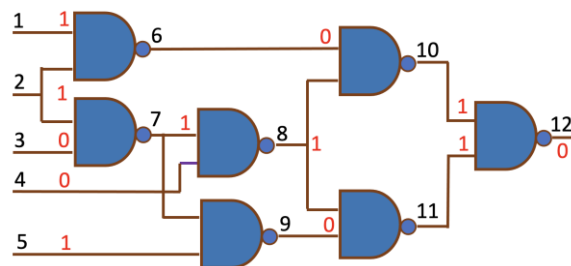


Figure 10. Circuit for simulation of faults

| No | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | NCI/NCO – number of primary circuit input/output=5/3 | | | | | | | | | | | | | |
|--------|---|---|---|---|---|-------------------|---|---|---|----|----|----|--|---|-----|---|-----------|----|---------------------|---|--------|----|----|--|--|--|
| 1 | 1 | | | | | | | | | | | | In | 1 | 2 | 3 | 4 | 5 | | | | | | | | |
| 2 | | 1 | | | | Simulation matrix | | | | | | | | | out | 8 | 10 | 12 | Circuit description | | | | | | | |
| 3 | | | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | 1 | | | | | | | | D-vectors | | | | L-vectors | | | | Inputs | | NI | | | |
| 6 | 1 | 1 | | | | 1 | | | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 2 | | | |
| 7 | | | 1 | | | | 1 | | | | | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 2 | 3 | 2 | | | |
| 8 | | | | 1 | | | | 1 | | | | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 7 | 4 | 2 | | | |
| 9 | | | 1 | | 1 | | 1 | | 1 | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 7 | 5 | 2 | | | |
| 10 | 1 | 1 | | | | 1 | | | | 1 | | | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 6 | 8 | 2 | | | |
| 11 | | | 1 | | 1 | | 1 | | 1 | | 1 | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 8 | 9 | 2 | | | |
| 12 | 1 | 1 | 1 | | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 10 | 11 | 2 | | | |
| Sum V | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Vector of External Circuit Outputs | | | | | | | | | | | | | |
| Good | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | G – Good behavior vector of the circuit lines | | | | | | | | | | | | | |
| Faults | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | F – detected faults on the input set | | | | | | | | | | | | | |

Figure 11. Data structure for circuit fault simulation on the input set 11001

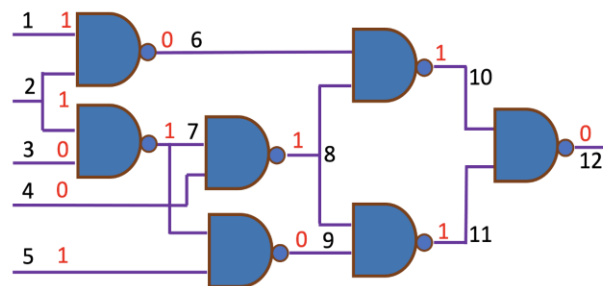
The circuit fault simulation metric consists of several key components and methodologies aimed at optimizing fault detection and diagnostics within a circuit system, as outlined in the following points. i) Smart and easy-to-implement structured data contains three macro components: circuit description, simulation matrix, and fault detection table. ii) A diagonal of 1 unit represents a simple fault model in a simulation matrix. iii) The simulation algorithm uses fault-as-addresses of the bits of deductive vectors composed from the input coordinates of the simulation table. iv) The algorithm for obtaining the deductive vector of an element uses only three operations on the logical vector and the binary input set. v) The simulation matrix structures the faults to be tested according to the external outputs of the circuit, which improves the quality of the test and provides additional opportunities to increase the depth of fault diagnostics during the operation of the device. vi) The structure of the simulation matrix makes it possible to handle faults of reconvergent fanouts as addresses without additional particular calculations. vii) Using the eventfulness principle in processing the simulation and deduction matrix makes it possible to increase the speed of fault analysis by 30%. Lastly, viii) when processing circuits with feedback, you must introduce pseudo-variables into the circuit model. When processing the simplest automata as elements, it is also necessary to enter pseudo-variables. Introducing pseudo-variables makes the fault simulation algorithm iterative when processing a binary test set. Figure 12 shows the interface for displaying information about the results of circuit fault simulations on the input binary test sets.

The circuit fault simulation table column components are i) input set, which is input binary test sets; ii) Q, which is the quality of the current test vector, defined as the number of faults to be tested on the test vector divided by twice the number of circuit lines $Q=F/2N$, iii) Σ or the integral quality of the test vectors, which is a non-additive assessment of the counting of the faults to be detected by the lines of the circuit or the columns of the table; if the test $T_{1...i,j}$ in the column j in lines 1... i tables are present 0.1, then the integral quality score is increased by two $\Sigma = \Sigma + 2$, If only 0 or 1 are present in the column, the integral score is increased by one $\Sigma = \Sigma + 1$ and the formula for calculating integral quality $\Sigma = \Sigma / 2N$; iv) 1...N, which are columns containing circuit line faults detected on test sets; and v) empty cells in the table that indicate the coordinates of the lines on which faults are not detected. The software implementation of the circuit fault simulation interface is shown in Figure 13. The output files of the circuit fault simulation interface are represented by three components (left to right). The first is the circuit fault simulation table on the whole test. If you click on a row in the table, a simulation matrix of the input set and data structure (logical and deductive vectors) appears. The second one is the schematic element simulation matrix on the input set. For visual verification of the simulation process, it is possible to track changes in data structures by generating fault vectors on each input binary set. Here are the last two lines: output (fault-free simulation vector) and (fault) fault detection vector. The third is deductive fault simulation vectors for circuit elements' input set and logic vectors.

| Table of test detected circuit fault | | | | | | | | | | | | | | |
|--------------------------------------|---------|--------|-----------------------------|---|---|---|---|---|---|---|---|----|----|----|
| Test sets | Quality | | Circuit line fault detected | | | | | | | | | | | |
| | Q set | Q test | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 00000 | 0.12 | 0.12 | | | | 1 | | | | 0 | | | | 0 |
| 00001 | 0.21 | 0.21 | | | | 1 | | 0 | | 0 | | 1 | | 0 |
| 00010 | 0.25 | 0.46 | | | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 00011 | 0.25 | 0.46 | | | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 00100 | 0.12 | 0.46 | | | | 1 | | | | 0 | | | | 0 |
| 00101 | 0.21 | 0.46 | | | | 1 | | 0 | | 0 | | 1 | | 0 |
| 00110 | 0.29 | 0.50 | | 1 | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 00111 | 0.29 | 0.50 | | 1 | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 01000 | 0.12 | 0.50 | | | | 1 | | | | 0 | | | | 0 |
| 01001 | 0.25 | 0.54 | 1 | | | 1 | | 0 | | 0 | | 1 | | 0 |
| 01010 | 0.29 | 0.58 | | | 1 | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 01011 | 0.29 | 0.58 | | | 1 | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 01100 | 0.08 | 0.58 | | | | | | | | 0 | | | | 0 |
| 01101 | 0.08 | 0.58 | | | | | | | | 0 | | | | 0 |
| 01110 | 0.21 | 0.71 | | 0 | 0 | | | | 1 | 0 | | | | 0 |
| 01111 | 0.21 | 0.71 | | 0 | 0 | | | | 1 | 0 | | | | 0 |
| 10000 | 0.12 | 0.71 | | | | 1 | | | | 0 | | | | 0 |
| 10001 | 0.25 | 0.71 | | 1 | | 1 | | 0 | | 0 | | 1 | | 0 |
| 10010 | 0.25 | 0.71 | | | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 10011 | 0.25 | 0.71 | | | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 10100 | 0.12 | 0.71 | | | | 1 | | | | 0 | | | | 0 |
| 10101 | 0.21 | 0.71 | | | | 1 | | 0 | | 0 | | 1 | | 0 |
| 10110 | 0.29 | 0.71 | | 1 | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 10111 | 0.29 | 0.71 | | 1 | | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 11000 | 0.25 | 0.83 | | | | 1 | 1 | | | 0 | 0 | | 1 | 0 |
| 11001 | 0.42 | 1.00 | 0 | 0 | 1 | | 0 | 1 | 0 | | 1 | 0 | 0 | 1 |
| 11010 | 0.29 | 1.00 | | | 1 | 0 | | | 0 | 1 | | 0 | 0 | 1 |
| 11011 | 0.21 | 1.00 | | | 1 | | | | 0 | | | 0 | 0 | 1 |
| 11100 | 0.17 | 1.00 | | | | | | | | 0 | 0 | | 1 | 0 |
| 11101 | 0.25 | 1.00 | | | 0 | | | | 1 | 0 | 0 | | 1 | 0 |
| 11110 | 0.29 | 1.00 | | 0 | 0 | | | | 1 | 0 | 0 | | 1 | 0 |
| 11111 | 0.29 | 1.00 | | 0 | 0 | | | | 1 | 0 | 0 | | 1 | 0 |

Figure 12. Circuit line fault simulation table

| Table of test detected circuit fault | | | | | | | | | | | | | | |
|--------------------------------------|------|------|---|---|---|---|---|---|---|---|---|----|----|----|
| | Q | I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 00000 | 0.12 | 0.12 | | | | | 1 | | | 0 | | | | 0 |
| 00001 | 0.21 | 0.21 | | | | | 1 | 0 | 0 | | | 1 | 0 | 0 |
| 00010 | 0.25 | 0.46 | | | | | 0 | 0 | 0 | 1 | | 0 | 0 | 1 |
| 00011 | 0.25 | 0.46 | | | | | 0 | 0 | 0 | 1 | | 0 | 0 | 0 |
| 00100 | 0.12 | 0.46 | | | | | 1 | 0 | | | | 0 | | 0 |
| 00101 | 0.21 | 0.46 | | | | | 1 | 0 | 0 | | | 1 | | 0 |
| 00110 | 0.29 | 0.50 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 00111 | 0.29 | 0.50 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 01000 | 0.12 | 0.50 | | | | | 1 | | | 0 | | | | 0 |
| 01001 | 0.25 | 0.54 | 1 | | | 1 | 0 | 0 | | | | 1 | | 0 |
| 01010 | 0.29 | 0.58 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 01011 | 0.29 | 0.58 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 01100 | 0.08 | 0.58 | | | | | | | | 0 | | | | 0 |
| 01101 | 0.08 | 0.58 | | | | | | | | 0 | | | | 0 |
| 01110 | 0.21 | 0.71 | | 0 | 0 | | | | 1 | 0 | | | | 0 |
| 01111 | 0.21 | 0.71 | | 0 | 0 | | | | 1 | 0 | | | | 0 |
| 10000 | 0.12 | 0.71 | | | | | 1 | | | 0 | | | | 0 |
| 10001 | 0.25 | 0.71 | | 1 | 1 | | 0 | 0 | | | 1 | | 0 | 0 |
| 10010 | 0.25 | 0.71 | | | | | 0 | 0 | 0 | 1 | | 0 | 0 | 1 |
| 10011 | 0.25 | 0.71 | | | | | 0 | 0 | 0 | 1 | | 0 | 0 | 1 |
| 10100 | 0.12 | 0.71 | | | | | 1 | | | 0 | | | | 0 |
| 10101 | 0.21 | 0.71 | | | | | 1 | 0 | 0 | | | 1 | | 0 |
| 10110 | 0.29 | 0.71 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 10111 | 0.29 | 0.71 | | 1 | 0 | | | 0 | 0 | 1 | | 0 | 0 | 1 |
| 11000 | 0.25 | 0.83 | | | | | 1 | 1 | | 0 | 0 | | | 1 |
| 11001 | 0.42 | 1.00 | 0 | 0 | 1 | | 0 | 0 | 1 | | 1 | 0 | 0 | 1 |
| 11010 | 0.29 | 1.00 | | | 1 | 0 | | | 0 | 0 | | 1 | 0 | 0 |
| 11011 | 0.21 | 1.00 | | | 1 | | | | 0 | | | 0 | 0 | 1 |
| 11100 | 0.17 | 1.00 | | | | | | | | 0 | 0 | | 1 | 0 |
| 11101 | 0.25 | 1.00 | | | 0 | | | | 1 | 0 | 0 | | 1 | 0 |
| 11110 | 0.29 | 1.00 | | 0 | 0 | | | | 1 | 0 | 0 | | 1 | 0 |
| 11111 | 0.29 | 1.00 | | 0 | 0 | | | | 1 | 0 | 0 | | 1 | 0 |



| Matrix of circuit fault simulation on input set | | | | | | | | | | | | Circiut data of input set 11001 | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|---------------------------------|----|----|----|----|----|----|----|----|----|----|
| BACK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | D0 | D1 | D2 | D3 | Q0 | Q1 | Q2 | Q3 | I0 | I1 |
| 1 | 1 | | | | | | | | | | | | | | | | | | | | | |
| 2 | | 1 | | | | | | | | | | | | | | | | | | | | |
| 3 | | | 1 | | | | | | | | | | | | | | | | | | | |
| 4 | | | | 1 | | | | | | | | | | | | | | | | | | |
| 5 | | | | | 1 | | | | | | | | | | | | | | | | | |
| 6 | | 1 | 1 | | | 1 | | | | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 |
| 7 | | | 1 | | | | 1 | | | | | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 2 | 3 |
| 8 | | | | 1 | | | | 1 | | | | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 7 | 4 |
| 9 | | | 1 | | 1 | | 1 | | 1 | | | | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 7 | 5 |
| 10 | 1 | 1 | | | | 1 | | | | 1 | | | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 6 | 8 |
| 11 | | | 1 | | 1 | | 1 | | 1 | | 1 | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 8 | 9 |
| 12 | 1 | 1 | 1 | | 1 | 1 | 1 | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 10 | 11 |
| output | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | | | | | | | | | | |
| fault | 0 | 0 | 1 | | 0 | 1 | 0 | | 1 | 0 | 0 | 1 | | | | | | | | | | |

Figure 13. Circuit fault simulation table

The following is a simulation of switch circuit faults on a complete verification test as shown in Figure 14. Adding external outputs (for example, 7, 8, 9) to the circuit affects the quality of the test and each test set. Making all the internal lines observable can significantly reduce the length of the test while keeping it 100% quality to detect all defects in the digital circuit. The inferences are described as follows. i) Faults as addresses simulation technique is proposed for processing input, internal, and output lines of a circuit based on logical vectors of functional elements, which are used to construct deductive vectors for transporting input faults to the circuit outputs. ii) Faults are treated as addresses for selecting the appropriate bits of deductive vectors, which makes it possible to increase the parallelism of circuit fault vector processing by increasing the complexity of logic elements. iii) The input fault model of the circuit on the input set is initially represented by a quadratic matrix of the number of the circuit line, where 1-values are diagonally arranged. iv) In the simulation process, such a matrix makes it easy to handle elements' most complex structural relationships, including reconvergence fanouts. v) The technique does not require synthesis to reduce logical functionalities to a specific basis of elements. Conversely, the technique will work faster if longer vectors in memory represent the macro functionalities. vi) The technique is focused on implementation in any memory based on the execution of read-write transactions, which makes it free from a potent CPU instruction system and economical in terms of energy and time for fault simulation. vii) To process circuits with feedback, you need to introduce pseudo-variables and additional loops to process the input binary vectors to bring the simulation results to a stable form. viii) To process digital automata, it is necessary to perform a preliminary synthesis of the automaton model to reduce it to a vector form with pseudo-variables. Python is the most popular function-oriented programming language [40], [41] today, which can process big data with the help of Excel files, including the simulation of large digital projects with the preservation of the result of such processing. Python supports operations on big data structures, which is especially important for implementing procedures and algorithms focused on in-memory computing. Therefore, smart data structures and fault-as-address simulation are implemented in Python code. The complexity of the program code for implementing FAAS-technique is estimated at 900 lines. The smart data structure and algorithms were verified on several dozen logic circuits on which fault detection tables were built.

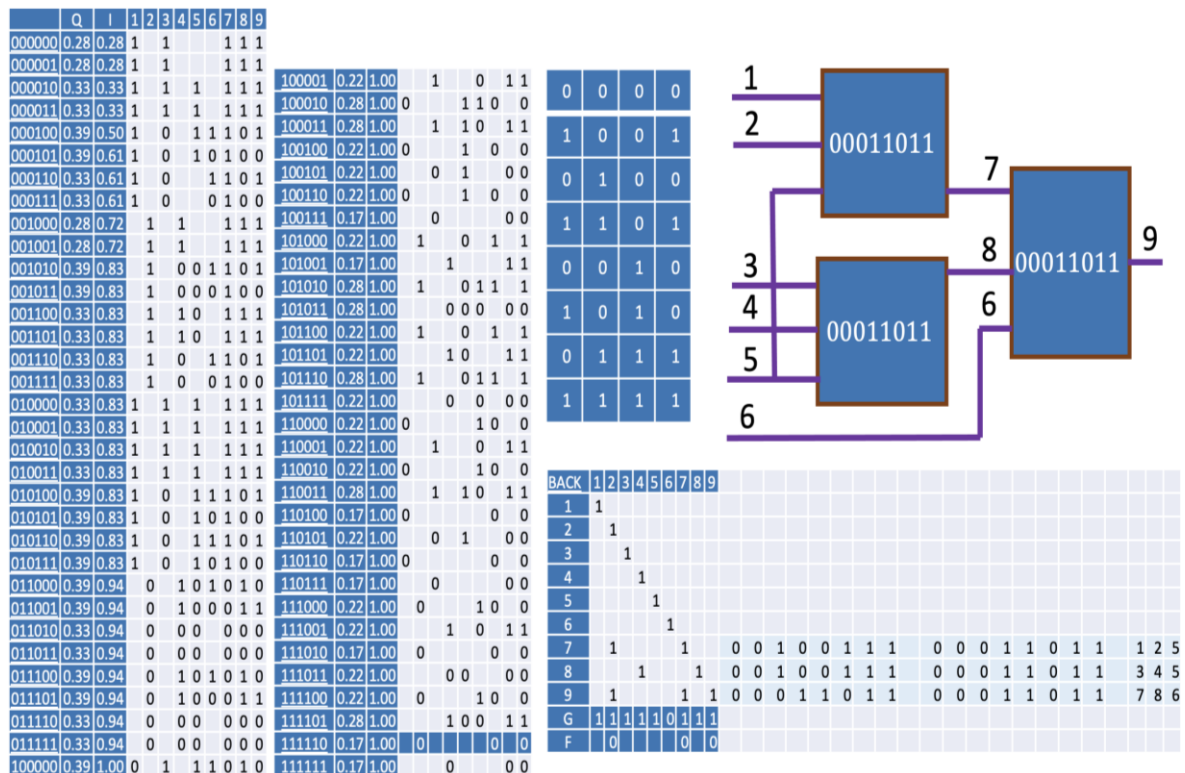


Figure 14. Result of simulation of switch circuit faults

6. CONCLUSION

The novelty of this is the future of the EDA market in implementing energy-saving in-memory design and test computing. FAAS is a technique that leverages input faults combination as the addresses of deductive vector bits that form the output vector of the detected input faults in the logic element. FAAS technique is proposed for digital circuit simulation, where logical vectors represent elements as a compact form of the truth table. The truth table is the longest-lived model of the computational process; it is more than 100 years old, and today, it is practically not used for organizing computations. Underneath the simplicity of the form of the truth table, which is understandable to humans and machines, there is an undeciphered genome of emerging computing. The truth table is proposed as an ideal data structure for fault simulation of input and internal lines of logic circuits. The logical vector in the truth table and the input test set are used to construct a deductive vector that transports any combination of faults in the circuit lines to its external outputs. Algorithms and data structures are proposed for in-memory simulating faults as addresses, as well as logical schemes of any structural complexity. Smart data structures have three macro components: circuit description, simulation matrix, and fault detection table. The schematic fault simulation algorithm metric contains the following novelty points: i) synthesis of a deductive vector on an input binary set to transport input faults to the output of an element; ii) development of a quadratic matrix of the circuit structure to simulate constant faults of input, internal, and output lines, including converging branches reconvergent fanouts; iii) simulation of circuit faults-as-addresses using a simulation matrix and deductive vectors of logic elements to obtain a vector of defects tested on the input test set; and iv) formation of a table of detected defects to determine the quality of the input sets and the test. Fault simulation algorithms use read-write transactions on smart data structures in any SoC, FPGA, ASIC, or RISC-V memory and do not require processor instructions and pre-synthesis of the circuit to bring logical functionalities to a specific element basis. The market goal of the proposed FAAS-Technique is to solve IP-core verification problems based on in-memory computing using the IEEE 1500 SECT standard. The data structure metric is logical vector or truth table. The metric of the simulation algorithms is the XOR transformation of the truth table on the test set $T \oplus L$ with the subsequent ordering of columns by address. The in-memory simulation uses read-write transactions, which makes the FAAS technique free of the CPU instruction system and cost-effective in terms of energy and latency to simulate fault-as-address.




REFERENCES

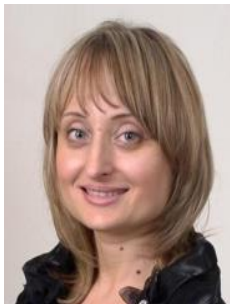
- [1] W. Gharibi, A. Hahanova, V. Hahanov, S. Chumachenko, E. Litvinova, and I. Hahanov, "Vector-deductive memory-based transactions for fault-as-address simulation," *Elektronnoe modelirovanie*, vol. 45, no. 1, pp. 3–26, Mar. 2023, doi: 10.15407/emodel.45.01.003.
- [2] A. Coluccio *et al.*, "Hybrid-SIMD: A modular and reconfigurable approach to beyond von Neumann computing," *IEEE Transactions on Computers*, pp. 1–1, 2021, doi: 10.1109/tc.2021.3127354.
- [3] 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.
- [4] B. Wu, H. Zhu, K. Chen, C. Yan, and W. Liu, "MLiM: High-performance magnetic logic in-memory scheme with unipolar switching SOT-MRAM," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 6, pp. 2412–2424, Jun. 2023, doi: 10.1109/tcsi.2023.3254607.
- [5] 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.
- [6] 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/AICAS54282.2022.9869991.
- [7] M. Moreau *et al.*, "Reliable ReRAM-based logic operations for computing in memory," Oct. 2018. doi: 10.1109/vlsi-soc.2018.8644780.
- [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] W. Gharibi, A. Hahanova, V. Hahanov, S. Chumachenko, E. Litvinova, and I. Hahanov, "Vector–logic synthesis of deductive matrices for fault simulation," *Elektronnoe modelirovanie*, vol. 45, no. 2, pp. 16–33, Apr. 2023, doi: 10.15407/emodel.45.02.016.
- [11] T. Liu, T. Yu, S. Wang, and S. Cai, "An efficient degraded deductive fault simulator for small-delay defects," *IEEE Access*, vol. 8, pp. 204855–204862, 2020, doi: 10.1109/access.2020.3037292.
- [12] M. Burhan *et al.*, "A comprehensive survey on the cooperation of fog computing paradigm-based IoT applications: layered architecture, real-time security issues, and solutions," *IEEE Access*, vol. 11, pp. 73303–73329, 2023, doi: 10.1109/access.2023.3294479.
- [13] Z. Kaya, M. Garrido, and J. Takala, "Memory-based FFT architecture with optimized number of multiplexers and memory usage," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 8, pp. 3084–3088, Aug. 2023, doi: 10.1109/tcsii.2023.3245823.




- [14] V. Hahanov *et al.*, “Vector-qubit models for SoC (system on a chip) logic-structure testing and fault simulation,” Feb. 2021. doi: 10.1109/cadsm52681.2021.9385266.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] Z. Zhao, P. X. Liu, and J. Gao, “Fault detection for non-Gaussian stochastic distribution systems based on randomized algorithms,” *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–9, 2022, doi: 10.1109/tim.2022.3192829.
- [19] V. Hahanov, *Cyber physical computing for IoT-driven services*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-54825-8.
- [20] R. Ubar, J. Raik, M. Jenihhin, and A. Jutman, *Structural decision diagrams in digital test: theory and applications*. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-44734-1.
- [21] E. J. Marinissen and Y. Zorian, “IEEE Std 1500 enables modular SoC testing,” *IEEE Design & Test of Computers*, vol. 26, no. 1, pp. 8–17, Jan. 2009, doi: 10.1109/mdt.2009.12.
- [22] K. Aslansefat and G.-R. Latif-Shabgahi, “A hierarchical approach for dynamic fault trees solution through semi-Markov process,” *IEEE Transactions on Reliability*, vol. 69, no. 3, pp. 986–1003, Sep. 2020, doi: 10.1109/TR.2019.2923893.
- [23] B. Kaczmarek *et al.*, “LBIST for automotive ICs with enhanced test generation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 7, pp. 2290–2300, Jul. 2022, doi: 10.1109/TCAD.2021.3100741.
- [24] S. Lee, K. Cho, S. Choi, and S. Kang, “A new logic topology-based scan chain stitching for test-power reduction,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3432–3436, Dec. 2020, doi: 10.1109/TCSII.2020.3004371.
- [25] P. Papavramidou and M. Nicolaidis, “Iterative diagnosis approach for ECC-based memory repair,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 2, pp. 464–477, Feb. 2020, doi: 10.1109/tcad.2018.2887052.
- [26] H. Zhang, K. Liu, M. Zhao, Z. Shen, X. Cai, and Z. Jia, “Pearl: Performance-aware wear leveling for nonvolatile FPGAs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 2, pp. 274–286, Feb. 2021, doi: 10.1109/tcad.2020.2998779.
- [27] D. V. Efanov, D. V. Pivovarov, and V. V. Khóroshev, “Conditions for detecting errors in the concurrent checking circuits by Boolean complement method to the code ‘2-out of-5,’” Nov. 2023. doi: 10.1109/ncs60404.2023.10397527.
- [28] A. Wagle and S. Vruthula, “Heterogeneous FPGA architecture using threshold logic gates for improved area, power, and performance,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 6, pp. 1855–1867, Jun. 2022, doi: 10.1109/tcad.2021.3099780.
- [29] L. Lin, Y. Huang, L. Xu, and S.-Y. Hsieh, “A complete fault tolerant method for extra fault diagnosability of alternating group graphs,” *IEEE Transactions on Reliability*, vol. 70, no. 3, pp. 957–969, Sep. 2021, doi: 10.1109/tr.2020.3021233.
- [30] I. Pomeranz, “Bit-complemented test data to replace the tail of a fault coverage curve,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 4, pp. 609–618, Apr. 2024, doi: 10.1109/tvlsi.2024.3365355.
- [31] D. Jinling and X. Aiqiang, “A fault simulation method based on mutated truth table of logic gates,” Nov. 2016. doi: 10.1109/icam.2016.7813557.
- [32] Q. Wang, T. Jin, and M. A. Mohamed, “A fast and robust fault section location method for power distribution systems considering multisource information,” *IEEE Systems Journal*, vol. 16, no. 2, pp. 1954–1964, Jun. 2022, doi: 10.1109/jsyst.2021.3057663.
- [33] J. Hu *et al.*, “Adaptive multidimensional parallel fault simulation framework on heterogeneous system,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1951–1964, Jun. 2023, doi: 10.1109/tcad.2022.3213617.
- [34] A. Ehteram, H. Sabaghian-Bidgoli, H. Ghasvari, and S. Hessabi, “A simple and fast solution for fault simulation using approximate parallel critical path tracing,” *Canadian Journal of Electrical and Computer Engineering*, vol. 43, no. 2, pp. 100–110, 2020, doi: 10.1109/cjece.2019.2950280.
- [35] 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.
- [36] A. Zarei and F. Safaei, “LIMITA: Logic-in-memory primitives for imprecise tolerant applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4686–4699, Nov. 2021, doi: 10.1109/tcsi.2021.3106017.
- [37] Z. Yao, Y. Liu, J. Chen, J. Ji, M. Zhang, and Y. Gong, “Active high-impedance fault detection method for resonant grounding distribution networks,” *IEEE Access*, vol. 12, pp. 10932–10945, 2024, doi: 10.1109/access.2024.3352258.
- [38] B. Wu, H. Zhu, K. Chen, C. Yan, and W. Liu, “MLiM: High-performance magnetic logic in-memory scheme with unipolar switching SOT-MRAMxx,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 6, pp. 2412–2424, Jun. 2023, doi: 10.1109/tcsi.2023.3254607.
- [39] 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.
- [40] C. Scalfani, “A new way to squash bugs: functional programming is hard to learn but yields fewer nasty surprises,” *IEEE Spectrum*, vol. 59, no. 12, pp. 40–45, Dec. 2022, doi: 10.1109/mspec.2022.9976475.
- [41] M. Zaman, K. Tanahashi, and S. Tanaka, “PyQUBO: Python library for mapping combinatorial optimization problems to QUBO form,” *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 838–850, Apr. 2022, doi: 10.1109/tc.2021.3063618.
- [42] 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, p. 274, Sep. 2023, doi: 10.11591/ijra.v12i3.pp274-288.

BIOGRAPHIES OF AUTHORS






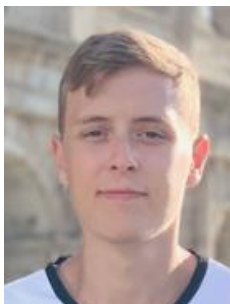
Vladimir Hahanov    was born in the USSR in 1953. He is a Doctor of Science, Professor of Computer Engineering Faculty, Design Automation Department, Kharkov National University of Radio Electronics, Ukraine. His research and development fields include design and test of computers, test generation and fault simulation for soc, quantum memory-driven computing, cyber-physical, cyber social computing, pattern recognition & machine learning computing, digital smart cyber university, and cloud-driven traffic control. He has supervised 4 Doctors of Science and 36 Ph.Ds. He has been the General Chair of the IEEE East-West Design & Test Symposium for 22 years since 2003. He is also the author of 650+ publications and 25 textbooks, 5 patents, and 202 Scopus-indexed papers: with 722 citations by 507 documents, h-index 14. Prof. Hahanov has been an IEEE Senior Member since 2010, IEEE Computer Society Golden Core Member, SAE member, and IFAC member, and communication society 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, 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 is also the author of 250+ publications, 10 textbooks, and 100 Scopus-indexed papers: with 325 citations, h-index 11. She can be contacted at svetachumachenko@icloud.com.






Eugenia Litvinova    was born in Kharkov 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 is also the author of 250+ publications 10 textbooks, and 100 Scopus-indexed papers: with 325 citations, h-index 11. She can be contacted at litvinova_eugenia@icloud.com.






Ivan Hahanov    was born in Kharkov in 1997. He is an IEEE member and a PhD in Computer Engineering, Kharkov National University of Radio Electronics, Ukraine. Ivan Hahanov is an Experienced Coursera-certified ML specialist and ex-mobile developer. He successfully accomplished and delivered projects using developed algorithms and acted as a team lead and the principal researcher on many of them. International activity: 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. Research fields: computer vision, machine learning, deep learning, data analysis, NLP, MLOps, cloud, and big data. Ivan Hahanov received a diploma as the best student in Ukraine in computer engineering and won the competition for the best scientific student work in Ukraine. He authorizes 50 publications, Scopus h-index: 5, 48 citations by 26 documents. Hobbies and interests: tennis, gymnastics, alpine skiing, tourism. He can be contacted at ivanhahanov@icloud.com or ivanhahanov@gmail.com.






Veronika Ponomarova    is an engineer. She was born in Kharkov in 1987. In 2012, she graduated from the Kharkov National University of Radio Electronics with a degree in computer systems and networks and received the qualification of a computer systems analyst. Research interests: design and test of computing, project management, and systems administration. Her hobbies are knitting, grooming. She can be contacted at veronika.ponomarova@nure.ua.



Hanna Khakhanova    was born in 1978 in Kharkov. Doctor of Science, Associate Professor of Design Automation Department, Computer Engineering Faculty, Kharkov National University of Radioelectronics, Ukraine. R&D fields: Cyber-physical, cyber-social computing, pattern recognition, and machine learning. Digital Smart Cyber University. 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. Previous positions: Deputy Dean of Computer Engineering Faculty (2013-2016). Author of more than 85 publications and 4 monographs, 1 patent, and 39 articles indexed in Scopus: 93 citations by 83 documents, h-index = 7. She can be contacted at anna.khakhanova@nure.ua or anna.hahanova@nure.ua.



Georgiy Kulak    was born in Kharkov in 2001. He is a Master of Computer Engineering. He was born in Kharkov in 2001. He is a Master of Computer Engineering, 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. He is the author of 4 publications, 1 patent, and 2 Scopus-indexed papers: with 3 citations by 2 documents. His dominating interests are programming and developing embedded devices. In his free time, he's making open-source tools and POCs, developing applications for IoT hardware and software for compact flying vehicles. His hobbies are gymnastics, paintball, karting, and running. He can be contacted at kulakgeorgij@gmail.com.