IAES International Journal of Robotics and Automation (IJRA)
Vol. 15, No. 2, June 2026, pp. 319~330
ISSN: 2722-2586, DOI: 10.11591/ijra.v15i2.pp319-330

Vector-logic models of digital circuits for simulation and rendering

Vladimir Hahanov, Svetlana Chumachenko, Eugenia Litvinova, Andrii Voronov, Oleh Demchenko, Nataliya Maksymova
Computer Engineering and Information Technology Faculty, Design Automation Department,
Kharkiv National University of Radio Electronics, Kharkiv, Ukraine

Article Info

Article history:

Received Feb 19, 2026
Revised Apr 12, 2026
Accepted Apr 19, 2026

Keywords:

Digital circuits
Fault simulation
Good-value simulation
Modelling for simulation
Rendering of inference
System on chip
Vector logic

ABSTRACT

Vector-logical in-memory computing for solving modelling for simulation (MOSI) problems by using read-write transactions free of processor instructions is proposed. A parser mechanism has been developed for converting the HDL code of the logical circuit into the internal vector-logical data structures of the MOSI service, addresses of logical vectors of elements. Deductive vectors are generated from the vector-logic model of the digital circuit for fault as address simulation of the input test sets. A mechanism for modelling a fault simulation matrix as the addresses of the deductive vector bits of each element on the test set has been created. The results of good-value and fault as address simulation are rendered and synchronized in the good-value simulation, fault as address simulation, fault simulation matrices on the input set, and on the lines of the logical circuit displayed on the monitor. Modeling and simulation mechanisms encoded and verified using examples of logical circuits of the ISCAS library. The scientific novelty is represented by vector-logical models of digital circuit elements, good-value simulation of test set as address and fault as address simulation of a digital circuit, and fault as address simulation of the input set, using a quadratic simulation matrix.

Corresponding Author:

Vladimir Hahanov
Kharkov National University of Radio Electronics,
Kharkiv, Ukraine
Email: vladimir.hahanov@nure.ua

1. INTRODUCTION

The purpose of a robot [1] is to replace a person in non-intellectual, monotonous tasks where humans tend to make mistakes [2]. At the same time, the robot's metrics [3] include long-term operational autonomy, minimal power consumption, operational accuracy in both time and space, and the reliability of control systems [4]. All these properties are effectively implemented in IP SoCs [5] that utilize energy-efficient, processorless in-memory computing architectures [5], [6] based on read-write transactions. The reliability of robot control systems is ensured by logical circuits [6] and simple, low-cost tools [7] for testing, modeling, and diagnostics based on vector logic. Why is this the preferred solution for the EDA market today? Industrial EDA [8] simulation systems (such as Synopsys, Sentaurus TCAD; Cadence, Virtuoso, Spectre, Voltus, Quantus, Xcelium, Allegro; Aldec, Active-HDL, Riviera-PRO; Siemens EDA ModelSim, Intel Quartus Prime, PSIM; Xilinx ISE, Vivado) are focused on helping testers [9] or designers verify [10] digital projects based on SoC, VLSI, ASIC, PLD, FPGA [11]. These systems cost tens of millions to hundreds of millions of dollars. They provide high-value simulation and fault-simulation services to generate tests and reduce test run times, which are crucial for large-scale digital projects. Such systems are generally

unavailable to small companies or most universities in the East or the West. Meanwhile, in the EDA domain of design and testing, fault simulation services are considered more than ten times as complex as typical simulation tools used to verify large projects with millions or billions of gates. However, it turns out that good-value simulation alone does not address testing issues for such projects. Consequently, standards [12] (IEEE 1500, P3405, 11.49, 1687) were developed to maintain SoC, ASIC, VLSI, and FPGA functionalities by dividing them into subcircuits, which are tested using the "divide and conquer" boundary-scan technology [13]. This approach allows the use of small testbenches to verify functions and structures as separate parts of a large project. More precise fault simulation mechanisms can then be employed to minimize testing time and diagnose defects. Therefore, segmenting large projects into smaller units based on function, structure, and an acceptable number of inputs enables the application of more accurate simulation methods [14]. After subdivision, each segment is processed using advanced fault-modeling techniques. This highlights the importance of innovative fault-modeling and testing-mapping methods for the functions and structures of large digital projects segmented into components. All current fault modeling methods use HDL languages for circuit description [15] (such as VHDL, Verilog, SystemVerilog, and SystemC), which depend on powerful processors [16] to generate compilation models [17] for large projects and to simulate defects. However, these compilation models are rigid logic that cannot be modified during testing. The data structures and algorithms involved are energy-intensive and difficult to encode. An alternative is to use simple, intelligent, explicit vector-logical data structures that significantly simplify analysis algorithms when modeling faults as truth-table addresses. Interpretive or explicit vector-logical models [18] are flexible structures that can be modified during verification and debugging [19]. When integrated into in-memory computing architectures [20], vector-logic mechanisms (models and algorithms) require only read-write transactions, eliminating the need for processor instructions [21]. This makes vector-logical computing [22] highly scalable and energy- and time-efficient for inference [23] Good-value simulation addresses whether the device operates correctly during exhaustive or limited testing [24]. It answers whether there are errors in the project: i) Fault simulation solves issues [25], [26]: i) Does the device work correctly on exhaustive testing? It offers a good-value simulation service; ii) Builds a fault testing map during exhaustive testing [27]; iii) Diagnose defects identified on each test. iv) Contributes to generating minimal tests for single constant faults; v) Develops schemes for seamless control of input variables affecting functionality, providing a Boolean derivative of input combinations; vi) Diagnoses defects in input variables of any order; vii) Defines test kits for checking combinations of faults at functionality inputs; and viii) All of these apply to both logical functions and any structures represented as graphs.

The study aims to develop vector-logical in-memory computing as a cost-effective solution for modeling and simulation (MOSI) problems using read-write transactions without processor instructions. Research objectives include: i) Developing a parser mechanism to convert HDL code of logical circuits into internal vector-logical data structures for the MOSI system; ii) Creating a vector-logical model of a digital circuit for efficient simulation of input test sets, represented as addresses of logical vector elements; iii) Building deductive vectors within a vector-logical model of a logical circuit for fault simulation based on address inputs; iv) Designing fault simulation matrices as bit addresses within the deductive vector of each element of the test set; v) Displaying and synchronizing the results of good-value and fault as address simulation, including tables and matrices, on the monitor screen; and vi) Developing and testing the modeling and simulation mechanisms on logical circuits from the ISCAS library [28], [29]. Scientific innovations include: i) Vector-logical models of circuit elements that enable good-value simulation of input test sets and fault simulation via bits of logical and deductive vectors based on read-write transactions; ii) Good-value simulation mechanisms that operate without processor instructions; iii) Fault simulation as an address-based mechanism functioning independently of a processor; iv) Fault address simulation of input sets using a quadratic simulation matrix; and v) Methods for rendering and synchronizing modeling results across four scalable windows displayed on the monitor. These groundbreaking innovations are unique worldwide and pave the way for cost-effective large-scale computing.

2. RENDERING VECTOR LOGIC SIMULATION OF A DIGITAL CIRCUIT

A rendering [30] of vector-logical modeling of a digital circuit is shown in Figure 1. The goal is to model the inputs and outputs for clear visualization of the processes and simulation results. In the figure, Tasks 1) is the display the HDL code of the digital circuit on the screen. Task 2) shows the good-value simulation mode of the circuit lines. Task 3) display the diagram of the line fault simulation table. Task 4) show the fault simulation matrix on the input data. Task 5) present on the screen vector-logical models of the digital circuit elements and the results of both good-value and fault simulation for all lines of the diagram. Ergonomic data structure rendering metrics: i) All windows for input data and simulation results are scalable from zero to the monitor size; ii) All simulation processes shown in windows 2-5 are synchronized with the input test exposure;

iii) Logical vectors represent all circuit elements, simplifying the modeling process to a read-write operation, similar to building with bricks, without processor instructions; iv) Visualizing input set quality and testing, along with good-value and fault detection on diagram lines, accelerates verification and the creation of minimal tests; v) Rendering the fault simulation matrix for the input set allows any student or engineer to learn the proposed vector logic modeling theory in just 10 minutes. This knowledge is crucial for identifying errors in digital device design and verifying testbenches. Industrial digital circuit simulation systems lack this feature; what's happening inside is hidden from the user and is expensive; and vi) A key feature and innovation of these services is the synchronization of modeling processes across all windows, illustrating all essential procedures to understand the results. This is very important for both engineers and students.

For those who appreciate vector-logical analysis, the following options are available: optional rendering for high-value and fault-simulation services. The first mode highlights circuit elements and wires by integrating the input set into the fault simulation matrix. This mode allows you to track the validating properties of the input good-value and fault-simulation test sets on all circuit lines. The second mode involves illuminating the logic element, its inputs, and outputs, in synchronization with the fixed line of the fault simulation matrix and the element's HDL code. A truth table is generated on the active circuit element output, where all combinations of stuck-at faults, which are covered by the unit coordinates of the deductive vector, are highlighted on its unit coordinates of binary addresses. The rows of the good-value and fault simulation tables are adjustable in this mode, enabling you to observe the good-value operation and fault simulation of the selected element across all test cases. The first mode is used to manually develop logic fault coverage tests, while the second mode helps precisely diagnose design errors, logic faults, or defects in a digital product.

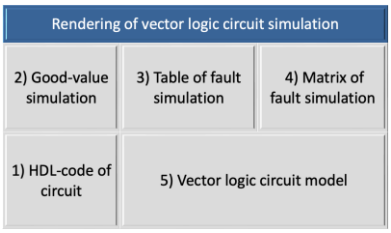


Figure 1. Vector logic modeling for rendering

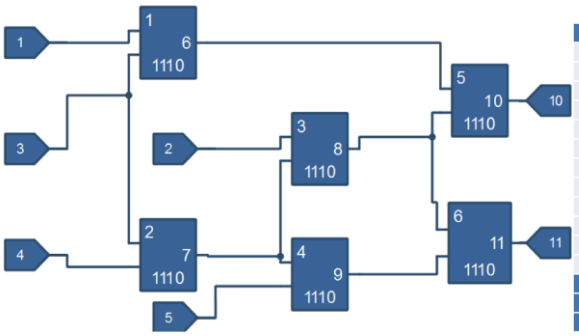
3. FAULT SIMULATION MATRIX

HDL (VHDL, Verilog, SystemVerilog, and SystemC) code for describing digital projects emerged in the EDA market in the 1980s [15]. Its emergence is associated with several key factors: i) The standard of communication between testers, designers, scientists, and students; ii) The standard for interoperability among various modeling systems and the design efforts of leading companies worldwide; iii) A benchmark for comparing existing EDA modeling tools in terms of simulation speed and accuracy; and iv) The organization of benchmark circuit libraries [31] featuring various circuit solutions [28] (ISCAS 85, 89, 96, 99) for testing existing and new design and verification methods. v) The development of powerful compilers capable of modeling large-scale digital projects. These factors are regularly considered by scientists and researchers in the EDA field to evaluate their results against existing benchmarks. Therefore, using a small C17 circuit from the ISCAS 85 library [29] effectively illustrates the operations and benefits of vector-logic modeling for simulation.

```
Verilog
module c17 (N1, N2, N3, N6, N7, N22, N23);
   input N1, N2, N3, N6, N7;
   output N22, N23;
   wire N10, N11, N16, N19;

   // Structural description on elements NAND
   nand NAND2_1 (N10, N1, N3);
   nand NAND2_2 (N11, N3, N6);
   nand NAND2_3 (N16, N2, N11);
   nand NAND2_4 (N19, N11, N7);
   nand NAND2_5 (N22, N10, N16);
   nand NAND2_6 (N23, N16, N19);
endmodule
```

This Verilog circuit description was used to obtain i) a monitor image of a vector logical model, and ii) a modeling of vector logic data structures – fault simulation matrix – for good-value as address and fault as address simulation on a test input set (see Figure 2).



Fault Simulation Matrix for 11111												Input	I-set	LV	DV
	1	2	3	4	5	6	7	8	9	10	11				
1	1														
2		1													
3			1												
4				1											
5					1										
6	1		1			1						1	3	1	1	1110	0111
7			1	1			1					3	4	1	1	1110	0111
8			1	1			1	1				2	7	1	0	1110	0100
9			1	1			1		1			7	5	0	1	1110	0010
10	1					1				1		6	8	0	1	1110	0010
11			1	1			1	1	1		1	8	9	1	1	1110	0111
sum	1		1	1		1	1	1	1	1	1	Date structure					
good	1	1	1	1	1	0	0	1	1	1	0						
fault	0		0	0		1	1	0	0	0	1						

Figure 2. Logic circuit and fault simulation matrix

The fault simulation matrix [22] is an intelligent simulation engine, or a smart vector-logic data structure, for efficiently simulating values and fault addresses across a test input set. Cleverly related here are the following components: i) A quadratic fault simulation matrix, initially filled with units at the coordinates of the main diagonal; ii) Date structure containing: Input – input line numbers of logic elements, I-set – input test sets for each circuit element, LV – logical vectors of each circuit element for good-value as address simulation, DV – deductive vectors of each circuit element for fault as address simulation; and iii) Inference of good-value as address and fault as address simulation results, the «sum» is an integral vector of logical or-summation of all detected faults on the observed outputs of the circuit. Two vectors or two matrix lines are summed up here, sum = 10∨11=10110111111, «good» – good-value as address simulation of all circuit lines, «fault» – fault as address simulation of all circuit line faults on the test input set.

A feature of fault simulation matrix is the unitary coding of all faults by unit coordinates of a quadratic matrix with dimension (n×n), where n is the number of input, output, and internal lines of the logical circuit. Each non-input row of the matrix is responsible for transporting faults of the input lines of the corresponding element to its output using the DV deductive vector, which is located in the same row of the matrix on the right. The address, which is formed by the one and zero coordinates of the simulation matrix at the input, selects a bit of the deductive vector that forms a (0)1 value at the output, which will mean (not) detected the combination of input faults, as the generated address, at the output of the element. For example, the sixth row of the matrix has 1-values at coordinates 1 and 3, which form the address 11 for the last bit of the deductive vector 0111. There is 1 at this address. This means that input faults at coordinates 1 and 3 will be detected at the output of element 6. The «sum» vector is a logical or-operation of all the vectors of the matrix that correspond to the observed output lines of the circuit. «Sum» row forms the «fault» vector of detected faults, which will always be the inverse of the good-value states of these lines in the «good» vector or on the test input set fault = (sum=1)&$\overline{good}$. Shown here is the date structure, which shows the formation of DV deductive vectors as a function of the input set and the logical vector of the LV element DV=f(I-set, LV). The deductive vector will be modeled in detail in the next section. The Fault Simulation Matrix is a body of know-how with no analogs worldwide, in terms of the simplicity of its data structures and the algorithm for processing them.

4. MODELING DEDUCTIVE VECTORS AND TESTING MAP

Vector-logical modeling of a digital circuit. The goal is to significantly reduce the time and energy costs of simulation modeling by using read-write in-memory computing transactions on vector-logical models of digital circuit elements. Vector logic modeling metric (Figure 3): i) A logical vector (01000010) is all that is needed to describe any functionality or structure; ii) A read-write transaction based on an explicit binary address of a logical vector bit is all that is needed to compute the state of any logical element; iii) Any two-seat operation is specified by a logical vector of 4 bits in length; iv) A two-place operation on a logical vector also generates a logical vector or matrix of length 2^{2n}; v) An Xor operation on a logical vector generates a disordered L-matrix of activity or a complete set of unordered deductive vectors and derivatives; vi) Ordering the activity matrix according to the HDL standard (H-matrix) by following the addresses of

logical variables generates a deductive D-matrix D = L_H. It contains the complete set of deductive vectors in rows and the complete set of derivatives in the columns of the matrix; vii) A deductive vector is a logical vector that determines the conditions for activating (changing) the output of an element by input logical faults on a given input set; viii) The Boolean derivative is a logical vector that determines the conditions for activating the output when a given number of input variables change on an exhaustive test; ix) C-matrix or testing map is a matrix of relationships between an exhaustive test and the complete set of all combinations of input variable faults. The coordinates of the matrix are determined by the F-vectors of the faults to be detected in alphabetical order (01), where the "dot" denotes a non-fault detected on the corresponding input variable. Each significant (0,1) coordinate of the F-vector is defined at the locations with 1-value of the address bits at the top of the C-matrix, which are further determined by the inverse value of the corresponding bits of the input test set on the left side of the matrix.

Three matrix operations are all that is needed to build a test map of any function or structure [23]. But our task is more difficult – to use vector logic to simulate the faults in all lines of a scheme built by someone else. To achieve this goal, we need to solve the following tasks (Figure 4): i) Create a parser for HDL code to describe the logical scheme into the internal data structures of MOSI services; ii) Modeling vector-logical data structures of a digital circuit for modeling and displaying on the screen; iii) Good-value simulation of test suites as addresses of bits of logical vectors of elements; iv) Simulation of faults as the addresses of the bits of logical vectors of elements; v) Rendering the good-value and fault simulation process on vector-logical data structures of the logical circuit using scrolling; vi) Rendering the vector-logical model of the scheme using scrolling and window scaling; vii) Rendering good-value and fault simulation on a logical schema image using scrolling and window scaling; viii) Rendering input set fault simulation for synthesizing a minimal logic circuit test; and ix) Modeling is the construction of a model of an object or phenomenon in the memory of a computer.

L-matrix								H-matrix								D-matrix								C-matrix or testing map							
	0	1	0	0	0	0	1	0									000	001	010	011	100	101	110	111		000	001	010	011	100	101	110	111
0		1					1		0	1	2	3	4	5	6	7	000		1				1		000		..1					11.	
1	1		1	1	1	1		1	1	0	3	2	5	4	7	6	001		1	1	1	1	1		001		..0	.1.	.10	1..	1.0	11.	
0		1					1		2	3	0	1	6	7	4	5	010				1	1			010				.01	1..			
0		1					1		3	2	1	0	7	6	5	4	011			1			1		011			.0.			1.0		
0		1					1		4	5	6	7	0	1	2	3	100			1			1		100			.1.			0.1		
0		1					1		5	4	7	6	1	0	3	2	101				1	1			101				.10	0..			
1	1		1	1	1	1		1	6	7	4	5	2	3	0	1	110		1	1	1	1	1		110		..1	.0.	.01	0..	0.1	00.	
0		1					1		7	6	5	4	3	2	1	0	111		1				1		111		..0					00.	

Figure 3. Matrix vector-logical modeling of the testing map

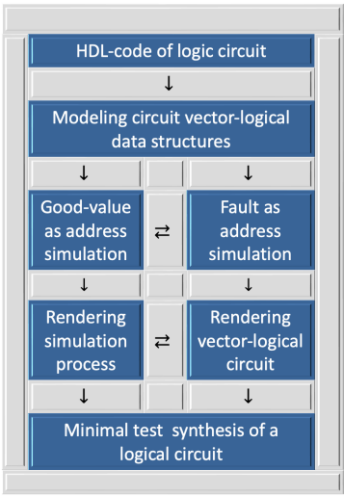


Figure 4. The map of the vector-logic circuit for simulation and rendering

Simulation involves using a model of a process or phenomenon to determine how it responds to specific inputs. Vector logic is the theory and practice of modeling data structures and algorithms to solve problems without a processor, using read-write transactions in computer memory. Rendering refers to the

optimization and ergonomic visualization of the relationship between an application's input and output data on the screen, aimed at entering the EDA market. An application map for verifying and rendering function structures and digital logic circuits is shown above. This kind of visualization for modeling and simulation processes is currently innovative and appealing not only to universities but also to specialists involved in testing modern SoC-based digital systems.

5. TRUTH TABLE MODELING DEDUCTIVE VECTOR

A mechanism for constructing a deductive vector on the input set using explicit addresses in the truth table is considered. This mechanism is easily scalable to any truth table from n variables. The mechanism can also be used to identify faults in test sets of any function or structure. Initial data (Figure 5): logical vector Y of functionality, test input set T.

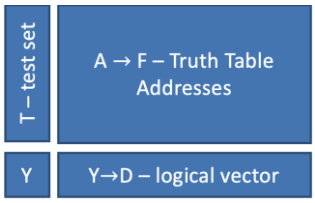


Figure 5. Data structures for modeling deductive vector

Algorithm of modeling deductive vector for the test set 011 for the logic vectors 01111110, 11100111 (Figure 6): i) The output function good-value state is calculated as the address of the truth table Y_T=f(F); iii) Vector xor operations L=TY⊕AY create an active unordered truth table on the test set; and iii) Ordering the resulting truth table according to the HDL address following standard A creates a deductive vector D_A=L in three operations.

Initial data for modeling: input binary set, length n: T=011 and logical vectors Y=01111110, 11100111, length 2^n, formative 2^n truth table input address F, on variables $x_1x_2x_3$. Output data: truth table of the faults being detected, dimension n× 2^n. The size of memory to store data structures is determined by the expression: M = n× 2^n + 2^n + n+1, where n is the number of variables of a logical element that forms compact and smart data structures: truth table F, logical (deductive) vector D, input binary set T=011, and output states Y=1, 0.

Modeling algorithm in example: In Model 1) Good-value simulation of logic element – computes the output value Y when feeding a binary input set T, treated as the address of the logic vector bit Y_T. Then, in Model 2), it executes the vector matrix operation A=T⊕F to obtain the activity truth table A. Here, the columns, the corresponding variables ($x_1x_2x_3$) treated as binary addresses 011, to recode or reorder the bits of a vector L. In Model 3), ordering L-Vector coordinates by input binary addresses $x_1x_2x_3$, and the truth tables for obtaining a deductive D vector D_A=L. This operation is necessary to bring the truth table to the generally accepted standard of incremented binary-decimal addresses based on input variables, which is required for processing elements in a digital structure. The computational complexity of generating a deductive vector is defined as $2^n \times (n+1)$.

Generation of detected input faults of a logic element by inverse values of input variable signals $x_1x_2x_3$ on the unit coordinates of the truth table columns covered (activated) by the 1-unit values of the deductive vector coordinates: $F_{ij} = \overline{T}_i \leftarrow F_{ij} \wedge D_i = 1$. The computational complexity of the algorithm is determined by the following formula: $Q = 1 + 2^n \times (n+1) + 2^n + 2^n \times n$ performing read-write transactions. It is proposed to use a truth table to simulate input faults for any logic element. The truth table for each input set gives a complete set of combinations of detected stuck-at faults ($x_2^0x_3^0 \vee x_1^1$), ($x_3^0 \vee x_2^0 \vee x_2^0x_3^0 \vee x_1^1 \vee x_1^1x_3^1 \vee x_1^1x_2^1$). In the backlight mode, a truth table is generated on the screen of the active circuit element at its output, where all combinations of stuck-at faults, which are covered by the unit coordinates of the deductive vector, are highlighted on its unit coordinates of binary addresses. Simulation formula: the input binary set used as the cell address of the logical vector contains the output state of the component that is concatenated with the input set to form a good state vector designed to XOR-interact with all columns of the truth table to create an unordered by addresses of the table and the activity vector to which the procedure of ordering the columns of the table and the bits of the activity vector by standard addresses in corresponding columns of the table, then the resulting deductive vector activates unit coordinates in the corresponding columns of the truth table with its unit values, the signs of which are determined by the inverse states of the input set bit.

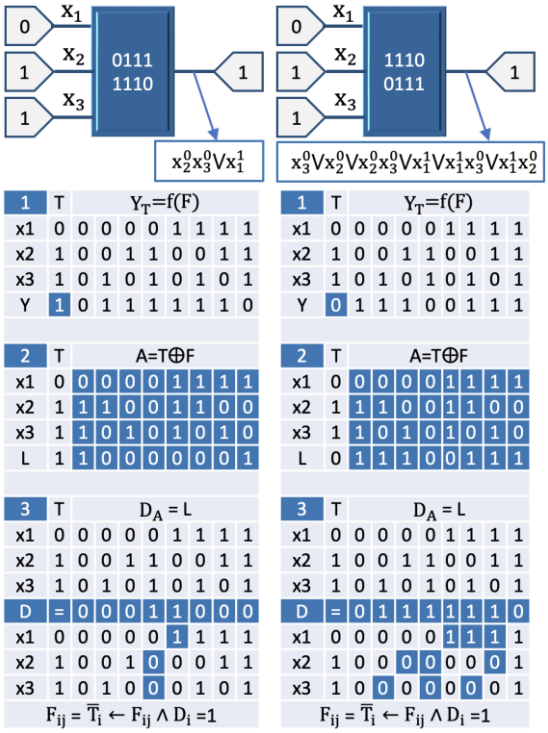


Figure 6. Modeling deductive vector on a test set 011 for the logic vectors 01111110, 1110011

6. GOOD-VALUE AND FAULT SIMULATION OF DIGITAL CIRCUIT

Vector logic simulation of digital circuits has the following metric: i) Extremely simple, smart, and redundant data structures that nullify the algorithm's good-value and fault simulation algorithms as address simulation; ii) In the extreme, these algorithms use read-write transactions without processor instructions when implemented in an in-memory computing architecture; iii) Rendering of simulation results is aimed at helping the tester solve problems of verifying functionalities or structures, minimizing tests, and diagnosing defects; iv) Synchronization of simulation processes across scalable windows shows all the details of the circuit simulation on a comprehensive test and on separate input sets, both on the diagram and on its individual elements; v) The windows contain the following statistical information: the time of execution of individual simulation procedures, the quality of test input sets Q, and the quality of the test Σ as a whole; vi) The circuit image on the monitor screen is synchronized with the fault simulation matrix and contains the logic fault check and good-value states of all circuit lines on each input set. All results from the good-value and fault simulation of the C17 circuit are shown in Figure 7 in two windows, synchronized by the dates of the simulation tables, the simulation matrix, and the circuit image on the monitor screen.

7. DISCUSSION OF THE RESULTS OF THE STUDY

The issues of using new technology to model input test sets as addresses of correct behavior and to represent faults as addresses in a logical vector are discussed. It employs an in-memory computing architecture based on read-write transactions to lower power consumption and reduce inference acquisition time. The modeling for simulation (MOSI) [31], [32] includes the following services: i) Modeling digital circuit data structures for accurate simulation; ii) Modeling data structures for simulation faults as addresses on test sets; and iii) Visualizing all processes involved in creating models of digital projects at the gate, register, and system levels for both accurate and fault simulation on input sets. These services can be used to verify digital logic projects and to study the mechanisms of vector logic modeling at technical universities.

The entire MOSI HDL program contains about 13,000 lines of code in the C++17 version, of which a small part is a web module written in JavaScript and HTML. The modeling module itself consists of more than 2,000 lines of code; the largest volume is occupied by the graphic module (view) – more than 8,000 lines. Windows and Ubuntu are supported. The structure of the MOSI HDL project, in terms of dataflow lines of code, is shown in Figure 8.

The modular structure of MOSI HDL uses the MVC (Model-View-Controller) design pattern. Each specific action (parsing, placement, tracing, table generation) is implemented as a separate command that interacts with the model, and the graphics display the model in various forms (tables, matrices, text, circuits).

The MOSI HDL program uses external libraries to solve problems not directly related to modeling: i) The open-source Surelog parser [33] with its universal hardware model UHDM is responsible for parsing Verilog files; ii) The GraphViz library is responsible for the placement of circuit elements (graph nodes) [34]; iii) The FLUTE package [35]–[37] is responsible for preliminary tracing of connections between elements (graph edges); iv) The final trace is performed by one of the search algorithms in the maze – A* search algorithm [38]; v) Graphic elements (in particular icons) are used from the Flaticon service [39]; vi) The graphical interface is written using the Qt library [40]. vii) Algorithms and other business logic are implemented using Boost [41]; vii) A web module is being developed separately, which, with the help of Docker containers and servers, allows you to host a desktop program as a web service; and viii) The following ISCAS85 circuits were processed to verify the modeling mechanisms: c17.v, c432.v, c499.v, c880.v, c1355.v, c1908.v, c3540.v, c6288.v. Circuit processing statistics as shown in Figure 9.

Test	Good-value simulation table											Test	Fault simulation table										
	1	2	3	4	5	6	7	8	9	10	11		Q	Σ	1	2	3	4	5	6	7	8	9	10	11
00000	0	0	0	0	0	1	1	1	1	0	0	00000	0.32	0.32		1			1	0		0	0	1	1
00001	0	0	0	0	1	1	1	1	0	0	1	00001	0.36	0.50		1			0	0	0	0	1	1	0
00010	0	0	0	1	0	1	1	1	1	0	0	00010	0.32	0.50		1			1	0		0	0	1	1
00011	0	0	0	1	1	1	1	1	0	0	1	00011	0.41	0.55		1	1		0	0	0	0	1	1	0
00100	0	0	1	0	0	1	1	1	1	0	0	00100	0.36	0.59	1	1			1	0		0	0	1	1
00101	0	0	1	0	1	1	1	1	0	0	1	00101	0.45	0.64	1	1		1	0	0	0	0	1	1	0
00110	0	0	1	1	0	1	0	1	1	0	0	00110	0.27	0.64	1					0		0	0	1	1
00111	0	0	1	1	1	1	0	1	1	0	0	00111	0.41	0.77	1		0	0		0	1	0	0	1	1
01000	0	1	0	0	0	1	1	0	1	1	1	01000	0.23	0.91		0					0	1		0	0
01001	0	1	0	0	1	1	1	0	0	1	1	01001	0.23	0.91		0					0	1		0	0
01010	0	1	0	1	0	1	1	0	1	1	1	01010	0.27	0.91		0	1				0	1		0	0
01011	0	1	0	1	1	1	1	0	0	1	1	01011	0.27	0.91		0	1				0	1		0	0
01100	0	1	1	0	0	1	1	0	1	1	1	01100	0.27	0.91		0		1			0	1		0	0
01101	0	1	1	0	1	1	1	0	0	1	1	01101	0.27	0.91		0		1			0	1		0	0
01110	0	1	1	1	0	1	0	1	1	0	0	01110	0.41	0.91	1		0	0		0	1	0	0	1	1
01111	0	1	1	1	1	1	0	1	1	0	0	01111	0.41	0.91	1		0	0		0	1	0	0	1	1
10000	1	0	0	0	0	1	1	1	1	0	0	10000	0.36	0.91		1	1		1	0		0	0	1	1
10001	1	0	0	0	1	1	1	1	0	0	1	10001	0.41	0.91		1	1		0	0	0	0	1	1	0
10010	1	0	0	1	0	1	1	1	1	0	0	10010	0.36	0.91		1	1		1	0		0	0	1	1
10011	1	0	0	1	1	1	1	1	0	0	1	10011	0.41	0.91		1	1		0	0	0	0	1	1	0
10100	1	0	1	0	0	0	1	1	1	1	0	10100	0.41	1.00	0	1	0		1	1		0	0	0	1
10101	1	0	1	0	1	0	1	1	0	1	1	10101	0.41	1.00	0		0	1	0	1	0		1	0	0
10110	1	0	1	1	0	0	0	1	1	1	0	10110	0.32	1.00	0		0			1		0	0	0	1
10111	1	0	1	1	1	0	0	1	1	1	0	10111	0.41	1.00	0		0	0		1	1	0	0	0	1
11000	1	1	0	0	0	1	1	0	1	1	1	11000	0.23	1.00		0					0	1		0	0
11001	1	1	0	0	1	1	1	0	0	1	1	11001	0.23	1.00		0					0	1		0	0
11010	1	1	0	1	0	1	1	0	1	1	1	11010	0.27	1.00		0	1				0	1		0	0
11011	1	1	0	1	1	1	1	0	0	1	1	11011	0.27	1.00		0	1				0	1		0	0
11100	1	1	1	0	0	0	1	0	1	1	1	11100	0.27	1.00		0		1			0	1		0	0
11101	1	1	1	0	1	0	1	0	0	1	1	11101	0.18	1.00				1			0			0	0
11110	1	1	1	1	0	0	0	1	1	1	0	11110	0.41	1.00	0		0	0		1	1	0	0	0	1
11111	1	1	1	1	1	0	0	1	1	1	0	11111	0.41	1.00	0		0	0		1	1	0	0	0	1

Figure 7. Inference of good-value and fault as addresses in the simulation

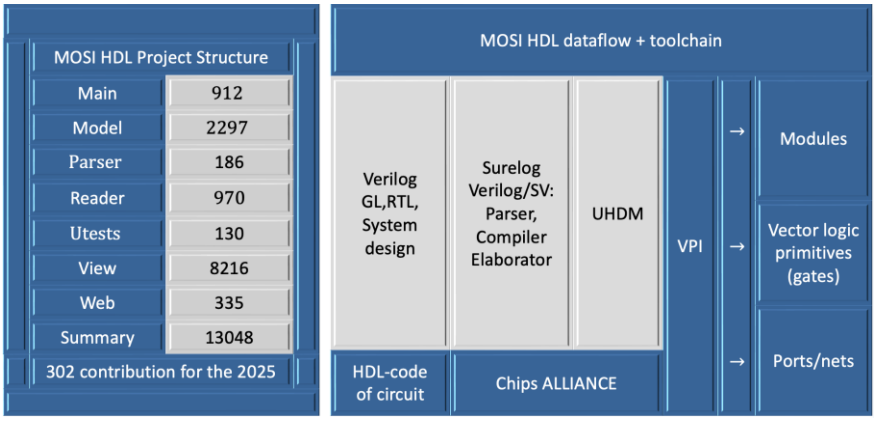


Figure 8. Project structure in lines of code and MOSI HDL dataflow

Actions	Circuits									
	Local Windows								Server Ubuntu	
	c17	c432*	c499*	c880*	c1355*	c1908*	c3540*	c6288*	c432*	c1355*
Verilog parsing using Surelog	0,343	1,3	1,4	2,8	4,7	4	9,7	20,3	1,6	4,5
Schematics placing using Graphviz	0,004	0,1	0,3	0,3	1,1	1,5	10,9	60,2	1,2	16,4
Schematics netline routing	0,003	0,5	0,8	0,6	1,8	4,6	10,3	4,5	0,6	1,8
Good value table generation	0,004	2,3	2,3	5,7	6,9	8,3	16,9	26,7	1,7	4,6
Fault simulation table generation	0,005	3,4	4,1	7,6	19,2	19	52	209,3	3,2	14,4
Total	0,359	7,7	8,8	17	33,7	37,4	99,9	320,9	8,3	41,8
Average fault matrix generation time	2E-04	0,003	0,004	0,007	0,019	0,019	0,051	0,204	0,003	0,014
** for 1024 vectors only*										

Figure 9. Latency of the modeling and simulation operation

Limitations of MOSI HDL: i) only Verilog GL, RTL, System combination code is supported without hierarchical support (one module without instantiating other modules) and only on scalar conductors; ii) the routing, although showing a high level of wiring (over 90%), is in many cases less than 100% and does not guarantee optimal results; iii) Circuits with more than 10 inputs in elements require manual input of test vectors; iv) development has been underway since May 2025. The repository contains 302 commits; each commit is preceded by 1-3 hours of development; and v) The total amount of coding work can be estimated at 500 hours.

8. CONCLUSION

Vector-logical in-memory computing is proposed as a cost-effective, low-budget solution for modeling and simulation (MOSI) problems that uses read-write transactions rather than processor instructions. A parser mechanism has been developed to convert the HDL code of the logical circuit into the MOSI service's internal vector-logical data structures. A vector-logical model of a digital circuit for efficient simulation of input test sets, using logical vector bit addresses, has been implemented. A mechanism for generating deductive vectors based on a vector-logical model of a digital circuit is proposed for fault-as-address simulation of input test sets. A mechanism for modeling a fault simulation matrix as the addresses of the deductive vector bits of each element on the test set has been created. The results of the good-value and fault-as-address simulations were rendered and synchronized in the tables for the good-value and fault-as-address simulations, the fault-modeling matrices on the input set, and the lines of the logical scheme displayed on the monitor screen. Modeling and simulation mechanisms are coded and verified using examples from the ISCAS library of logical circuits. The processing time of logic circuits and mechanisms for fault modeling and good-value simulation is presented. The scientific novelty of the study lies in vector-logical models of digital circuit elements, high-value simulation of the test set, address- and fault-as-address simulation of the digital circuit, and fault-as-address simulation of the input set, all using a quadratic simulation matrix. Implemented mechanisms for ergonomic rendering and synchronization of modeling results in four scalable windows displayed on the monitor screen. All of these innovative solutions are unparalleled worldwide and create a cost-effective future of mass computing. MOSI cloud services can be useful to students and engineers engaged in verifying digital projects based on vector logic elements.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Vladimir Hahanov	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Svetlana Chumachenko					✓	✓			✓	✓		✓	✓	
Eugenia Litvinova						✓			✓	✓				
Andrii Voronov			✓			✓				✓				
Oleh Demchenko			✓			✓				✓				
Nataliya Maksymova				✓		✓				✓				

C : **C**onceptualization	I : **I**nvestigation	Vi : **Vi**sualization
M : **M**ethodology	R : **R**esources	Su : **Su**pervision
So : **So**ftware	D : **D**ata Curation	P : **P**roject administration
Va : **Va**lidation	O : Writing - **O**riginal Draft	Fu : **Fu**nding acquisition
Fo : **Fo**rmal analysis	E : Writing - Review & **E**diting	

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

- The data that support the findings of this study are available on request from the corresponding author, [Vladimir Hahanov].
- Derived data supporting the findings of this study are available from the corresponding author [Vladimir Hahanov] on request.
- The authors confirm that the data supporting the findings of this study are available within the article.

REFERENCES

[1] S. Kim *et al.*, "MAVERIC: A 16nm 72 FPS, 10 mJ/frame heterogeneous robotics SoC with 4 Cores and 13 INT8/FP32 Accelerators," in *Digest of Technical Papers - Symposium on VLSI Technology*, 2025, pp. 1–3. doi: 10.23919/VLSITechnologyandCir65189.2025.11075163.

[2] R. R. Chekuri, V. Shivaram, K. Shravan, T. V. Sai, M. C. Chinnaiah, and D. H. Krishna, "Design and implementation of line follower robot on SoC with SDK," in *2024 3rd International Conference for Innovation in Technology, INOCON 2024*, 2024, pp. 1–5. doi: 10.1109/INOCON60754.2024.10511509.

[3] D. Jiang *et al.*, "Live demonstration: a reconfigurable, energy-efficient and high-frame-rate EKF-SLAM accelerator based SoC design for autonomous mobile robot applications," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2024, p. 1. doi: 10.1109/ISCAS58744.2024.10558139.

[4] J. Lee and J. H. Yoon, "A neuromorphic SLAM Accelerator supporting multi-agent error correction in swarm robotics," in *Proceedings - International SoC Design Conference 2022, ISOCC 2022*, 2022, pp. 241–242. doi: 10.1109/ISOCC56007.2022.10031388.

[5] S. Mahmood Mahdi, A. I. Abdulkareem, A. Jaleel Humaidi, A. K. Al Mhdawi, and H. Al-Raweshidy, "Comprehensive review of control techniques for various mechanisms of parallel robots," *IEEE Access*, vol. 13, pp. 63381–63416, 2025, doi: 10.1109/ACCESS.2025.3557937.

[6] K. Deng *et al.*, "System-on-chip test and characterization: a review," *IEEE Transactions on Instrumentation and Measurement*, vol. 74, pp. 1–28, 2025, doi: 10.1109/TIM.2025.3584134.

[7] B. S. I. S. Publication, "BSI Standards publication quality of electronic and software intellectual property used in system and system on chip (SoC) Designs," *IEC 62014-5 IEEE Std 1734-2011*, pp. 1–42, Mar. 2015, doi: 10.1109/IEEESTD.2015.7067346.

[8] J. Sifakis, "System design automation: challenges and limitations," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2093–2103, Nov. 2015, doi: 10.1109/JPROC.2015.2484060.

[9] B. Zhao *et al.*, "EDA-Q: electronic design automation for superconducting quantum chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 45, no. 1, pp. 266–279, Jan. 2026, doi: 10.1109/TCAD.2025.3580341.

[10] H. Du, X. Du, and J. Yang, "ASGF4EPD: An automated schematic generation framework for electronic products design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026, doi: 10.1109/TCAD.2026.3652497.

[11] S. Srilakshmi and G. L. Madhumati, "A comparative analysis of HDL and HLS for developing CNN accelerators," in *Proceedings of the 3rd International Conference on Artificial Intelligence and Smart Energy, ICAIS 2023*, 2023, pp. 1060–1065. doi: 10.1109/ICAIS56108.2023.10073746.

[12] K. J. Lee, T. Y. Hsieh, C. Y. Chang, Y. T. Hong, and W. C. Huang, "On-chip SOC test platform design based on IEEE 1500 standard," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1134–1139, Jul. 2010, doi: 10.1109/TVLSI.2009.2019978.

[13] F. Restuccia and R. Kastner, "Cut and Forward: safe and secure communication for FPGA System on Chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4052–4063, Nov. 2022, doi: 10.1109/TCAD.2022.3197343.

[14] K. J. Lee, T. Y. Hsieh, C. Y. Chang, Y. T. Hong, and W. C. Huang, "On-chip SOC test platform design based on IEEE 1500 standardd," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1134–1139, Jul. 2010, doi: 10.1109/TVLSI.2009.2019978.

[15] D. Kovachev, S. Belchev, D. Nedeva, and T. Uzuntonev, "Some guidelines on HDL selection for digital systems design," in *Proceedings of the International Conference on Biomedical Innovations and Applications, BIA 2021*, 2021, pp. 77–80. doi: 10.1109/BIA52594.2022.9831236.

[16] S. Froehlich and R. Drechsler, "LiM-HDL: HDL-Based synthesis for in-memory computing," in *Proceedings of the 2022 Design, Automation and Test in Europe Conference and Exhibition, DATE 2022*, 2022, pp. 1395–1400. doi: 10.23919/DATE54114.2022.9774627.

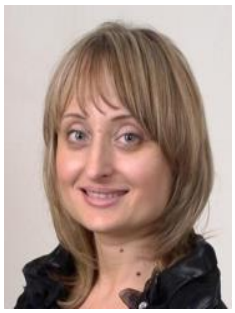
[17] S. Nakamura, H. Nakayama, R. Onuma, J. Tachibana, H. Kaminaga, and Y. Miyadera, "A system for identification of stumbles in construction of program logic that does not appear as compilation errors," in *2022 IEEE International Conference on Computing, ICOCO 2022*, 2022, pp. 43–48. doi: 10.1109/ICOCO56118.2022.10031920.

[18] Z. Y. Xiong and Z. X. Yan, "Research on audio playing system design factors modeling based on interpretive structure model," in *Proceedings - 2021 International Conference on Machine Learning and Intelligent Systems Engineering, MLISE 2021*, 2021, pp. 33–37. doi: 10.1109/MLISE54096.2021.00014.

[19] N. Pouti, "Interpretive structural modeling of uncertainty reduction factors in social commerce," in *International Conference on Web Research, ICWR*, 2025, no. 2025, pp. 88–99. doi: 10.1109/ICWR65219.2025.11006245.
[20] N. Darabi, P. Shukla, D. Jayasuriya, D. Kumar, A. C. Stutts, and A. R. Trivedi, "Navigating the unknown: uncertainty-aware compute-in-memory autonomy of edge robotics," in *Proceedings -Design, Automation and Test in Europe, DATE*, 2024, pp. 1–6. doi: 10.23919/date58400.2024.10546628.
[21] N. R. Shanbhag and S. K. Roy, "Benchmarking in-memory computing architectures," *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 288–300, 2022, doi: 10.1109/OJSSCS.2022.3210152.
[22] V. Hahanov, S. Chumachenko, E. Litvinova, I. Hahanov, Z. Davitadze, and D. Devadze, "Vector logic of computing," in *2025 IEEE 13th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2026, pp. 1–5. doi: 10.1109/idaacs68557.2025.11322309.
[23] V. Hahanov, W. Gharibi, S. Chumachenko, and E. Litvinova, "Vector synthesis of fault testing map for logic," *IAES International Journal of Robotics and Automation*, vol. 13, no. 3, pp. 293–306, 2024, doi: 10.11591/ijra.v13i3.pp293-306.
[24] T. Sharma, I. Chakraborty, M. Ali, and K. Roy, "Evaluating compute in memory architectures for matrix multiplication: a dataflow-centric perspective," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2025, pp. 1–3. doi: 10.1109/ispass64960.2025.00056.
[25] H. Guo, J. Chen, and H. Jiao, "A 15T SRAM cell-based fully-digital computing-in-memory macro supporting high parallelism and fine-grained simultaneous Read + Write + MAC operations," *IEEE Solid-State Circuits Letters*, vol. 8, pp. 153–156, 2025, doi: 10.1109/LSSC.2025.3567840.
[26] S. Zhang, X. Cui, F. Wei, and X. Cui, "An area-efficient in-memory implementation method of arbitrary Boolean function based on SRAM array," *IEEE Transactions on Computers*, vol. 72, no. 12, pp. 3416–3430, Dec. 2023, doi: 10.1109/TC.2023.3301156.
[27] H. Lee, J. Lee, and S. Kang, "An efficient test architecture using hybrid built-in self-test for processing-in-memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 33, no. 5, pp. 1452–1456, May 2025, doi: 10.1109/TVLSI.2024.3504539.
[28] S. Kajihara, K. Kinoshita, I. Pomeranz, and S. M. Reddy, "Combinationally redundant ISCAS-89 benchmark circuits," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 1996, vol. 4, pp. 632–634. doi: 10.1109/iscas.1996.542103.
[29] D. Bryan, "ISCAS benchmark circuits overview." Accessed: Feb. 09, 2026. [Online]. Available: https://ru.scribd.com/doc/164149583/Iscas-85
[30] C. Yang and B. Li, "Research on optimization strategies for gpu cluster cloud rendering virtual simulation resources," in *Proceedings - 2024 14th International Conference on Information Technology in Medicine and Education, ITME 2024*, 2024, pp. 392–395. doi: 10.1109/ITME63426.2024.00084.
[31] "MOSI HDL." Accessed: Feb. 09, 2026. [Online]. Available: www.mosihdl.work
[32] "Vector Simulation Model." Accessed: Feb. 09, 2026. [Online]. Available: https://vector-simulation-model.vercel.app/
[33] Alain Dargelas, "Surelog," *Git*. Accessed: Feb. 09, 2026. [Online]. Available: https://github.com/chipsalliance/Surelog
[34] "Graphviz - Graph visualization and layout algorithms." https://mosilogic.github.io/genlv (accessed Feb. 09, 2026).
[35] C. Chu and Y. C. Wong, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, Jan. 2008, doi: 10.1109/TCAD.2007.907068.
[36] "FLUTE (The OpenROAD Project Attic)," *GitHub*. Accessed: Feb. 09, 2026. [Online]. Available: https://github.com/The-OpenROAD-Project-Attic/flute/
[37] T. Huang, L. Li, and E. F. Y. Young, "On the construction of optimal obstacle-avoiding rectilinear steiner minimum trees," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 718–731, May 2011, doi: 10.1109/TCAD.2010.2098930.
[38] Z. Jiang, T. Wang, and J. Yan, "Unifying offline and online multi-graph matching via finding shortest paths on supergraph," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3648--3663, 2020, doi: 10.1109/TPAMI.2020.2989928.
[39] "Flaticon - Resources made by and for designers." Accessed: Feb. 09, 2026. [Online]. Available: https://www.flaticon.com/
[40] "Qt - Cross-platform application development framework." https://www.qt.io/ (accessed Feb. 09, 2026).
[41] "Boost - Portable, peer-reviewed C++ libraries." https://www.boost.org/ (accessed Feb. 09, 2026).

BIOGRAPHIES OF AUTHORS

Vladimir Hahanov was born in USSR in 1953. He is a Doctor of Science, Professor of Computer Engineering and Information Technology 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 and vector-logic computing, cyber-physical and 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 Ph.Ds. He has been the General Chair of the IEEE East-West Design & Test Symposium for 23 years since 2003. He is also the author of 650+ publications and 25 textbooks, 5 patents, and 216 Scopus-indexed papers, with 1240 citations by 803 documents. H-index 16. Prof. Hahanov has been an IEEE Senior Member since 2010, an IEEE Computer Society Golden Core Member, an SAE member, an IFAC member, and a Communication Society member. He can be contacted at vladimir.hahanov@nure.ua or 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, Computer Engineering and Information Technology Faculty, Kharkov National University of Radio Electronics, Ukraine. Her research and development fields include Mathematics, Computer Engineering, and Smart Cyber University. Her international activities include fundamental research under cooperation agreements on scientific and technical "Strategic Partnership" with Aldec Inc. (USA) in 2000 and 2005; SEIDA BAITSE "Baltic Academic IT Security Exchange," Blekinge Institute of Technology, Sweden in 2011–2012; and the 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 and 5 textbooks, and 112 Scopus-indexed papers, with 561 citations by 414 documents. H-index 12. She can be contacted at svetlana.chumachenko@nure.ua.

Eugenia Litvinova was born in Ukraine in 1962. She is a Doctor of Science and a Professor of Computer Engineering and Information Technology 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 serving as 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 as a member of the organizing committee of the IEEE East-West Design & Test Symposium from 2007 to the present. She is also the author of 250+ publications and 5 textbooks, 3 patents, and 119 Scopus-indexed papers, with 637 citations by 468 documents. H-index 13. She can be contacted at eugenia.litvinova@nure.ua or litvinova_eugenia@icloud.com.

Andrii Voronov is a Ph.D. student in the Design Automation Department, Computer Engineering and Information Technology Faculty, Kharkov National University of Radio Electronics, Ukraine. His research and development fields include cyber-physical computing, SoC design and test, and machine learning. He can be contacted at andrii.voronov@nure.ua.

Oleh Demchenko is a Ph.D. student in the Design Automation Department, Computer Engineering and Information Technology Faculty, Kharkov National University of Radio Electronics, Ukraine. His research and development fields include cyber-physical computing, SoC design and test, and machine learning. He can be contacted at oleh.demchenko@nure.ua.

Nataliya Maksymova was born in Ukraine. She holds a master's degree in computer science from Kharkov National University of Radio Electronics, Ukraine, evaluated by the University of Toronto as equivalent to a Canadian master's degree. Nataliya Maksymova is a Professor at Niagara University, Vaughan Campus, Ontario, Canada. Her research and development fields include SoC design and test and machine learning. She can be contacted at nmaximova@hotmail.com.