Models and Methods for Verification and Diagnosis of SoC HDL-code

Vladimir Hahanov, *Senior Member, IEEE* , Wajeb Gharibi, Eugenia Litvinova, *Member, IEEE*, Svetlana Chumachenko, *Member, IEEE*

Abstract — **Xor-metrix for object relations in a vector logic space and a structural testing model are proposed. Assertion-based models and methods for the verification and diagnosis of HDL-code functional failures, which make possible to reduce considerably time-to-market of software and hardware, are developed. An architectural model of multimatrix reduced logical instruction set processor for embedded diagnosing is offered.**

I. Introduction

Recent trends in creating new communications, computing and information services, useful to the human, are development of dedicated gadgets, which have important advantages over PCs and laptops: power consumption, compactness, weight, cost, functionality, and friendliness of interface. Practically the top ten dedicated products 2010 (Apple iPad, Samsung Galaxy S, Apple MacBook Air, Logitech Revue, Google Nexus One (HTC Desire), Apple iPhone 4, Apple TV, Toshiba Libretto W100, Microsoft Kinect, Nook Color) is realized as digital systems-on-chips. By 2012 the mobile and wireless communication market will move to 20 nm (results of the January 2011 Technology Forum of Common Platform Alliance). Further development of the technologies by year: 2014 – 14 nm, 2016 – 11 nm. In 2015 more than 55% of mobile phones will be smartphones, tablet PCs will replace laptops and netbooks. Superfones (Nexus-1, Google) will unite all devices and services. The transition from the computing platform to mobile devices with small size results in considerable reduction in power consumption worldwide. The next computerization wave, entitled "Internet of things", is being accelerated. It will lead to widespread sensor networks, including their integration into the human body. The world market of the above devices and gadgets today involves about 3 billion products. For their effective designing, manufacturing and exploitation the new technologies and Infrastructures IP are created. One of the possible steps in this direction is represented below in the form of verification technology T^v: M^t is metrics and model for testing, H^c is HDL-code of a design, G^t is synthesis of software transaction graph, $\{M^f, M^s\}$ determine creating two verification models for HDL-code (functional failure table and software activation matrix), $\{D^c, D^r, D^m\}$ determine developing three methods for diagnosing the functional failures (for analyzing rows, columns and whole matrix), which use the assertion engine (assertion is a logical statement for detecting the semantic errors in software), P^m is architecture of multimatrix processor for parallel analyzing tabular data, R is implementation of models, methods and tools in the system Riviera, Aldec Inc.:

$$T^v = M^t \to H^c \to G^t \to \left\{ \begin{matrix} M^f \to \left\{ \begin{matrix} D^c \\ D^r \end{matrix} \right. \\ M^s \to D^m \end{matrix} \right\} \to P^m \to R.$$

The objective of the research is to reduce time-to-market and improve the quality of digital systems-on-chips by developing the assertion-based infrastructure, models and methods for verification and diagnosis HDL-code. The information, needed for detecting failures at the functional blocks, is formed during simulation (execution) of software code. Design effectiveness for digital product is determined as the average and normalized in the range [0,1] integral criterion:

$$E = F(L,T,H) = \min[\frac{1}{3}(L+T+H)],\ Y = (1-P)^n;$$

$$L = 1 - Y^{(1-k)} = 1 - (1-P)^{n(1-k)};$$

$$T = [(1-k) \times H^s]/(H^s + H^a);\ H = H^a/(H^s + H^a).$$

The criterion takes into account the following: the error level L, the verification time T, software-hardware redundancy, determined by the assertion engine and Infrastructure IP tools H. The parameter L, as a complement of the parameter Y (yield), depends on the testability k of a

Manuscript received December 3, 2010.

Vladimir Hahanov is with the Kharkov National University of Radioelectronics, Ukraine, 61166, Kharkov, Lenin Prosp., 14, room 321 (corresponding author to provide phone: (057)7021326; fax: (057)7021326; e-mail: hahanov@ kture.kharkov.ua).

Wajeb Gharibi is with Jazan University, P. O. Box 4425 Arrawabi, Unit #1, Jazan 82822-6694, KSA. Mobile: +966 508 2232 64, E-mail: gharibi@jazanu.edu.sa, gharibiw@hotmail.comz

Eugenia Litvinova is with the Kharkov National University of Radioelectronics, Ukraine, 61166, Kharkov, Lenin Prosp., 14, room 378 (phone: (057)7021326; fax: (057)7021326; e-mail: ri@ kture.kharkov.ua).

Chumachenko Svetlana is with the Kharkov National University of Radioelectronics, Ukraine, 61166, Kharkov, Lenin Prosp., 14, room 321 (phone: (057)7021326; fax: (057)7021326; e-mail: ri@ kture.kharkov.ua).

design, the probability P of existence of faulty components, and the quantity of undetected errors n. The time of verification is determined by the testability of a design k [3,4], multiplied by the structural complexity of hardware-software functionality, divided by the total complexity of a design in code lines. The software-hardware redundancy depends on the complexity of assertion code and other costs, divided by the total design complexity. At that software or hardware redundancy has to provide the specified diagnosis depth for functional errors and time-to-market, defined by customer.

The problems are: 1) Creation of a metrics and structural-analytical model for testing digital systems-on-chips. 2) Improvement of the models and methods for detecting functional failures, based on assertion engine, to increase the speed of HDL-code verification and diagnosis. 3) Development of the architectural model of multimatrix processor for diagnosing.

References are: 1. Models of the problems for technical diagnosis are presented in [1-6]. 2. Diagnosis and verification of digital systems-on-chips are described in [9-17, 22-15]. 3. Hardware and matrix processors for increasing the speed of testing are proposed in [18-21].

II. A MODEL FOR TESTING AND VERIFICATION

The effective process models and methods for diagnosing the functional failures in software and/or hardware are offered. The register or matrix (tabular) data structures, focused to parallel execution of logic operations, are used for detecting the faulty components.

The problem of synthesis or analysis of system components can be formulated in the form of interaction (symmetrical difference is an analog of xor-operation on the Boolean) of its model F, input stimuli T and responses L in a cybernetic space:

$$f(F,T,L)=\varnothing \rightarrow F\Delta T\Delta L=\varnothing .$$

A cyberspace is a set of information processes and occurrences, which use computer systems and networks as a carrier. Particularly, a space component is represented by k-dimensional (tuple) vector $a=(a_1,a_2,...,a_j,...,a_k)$, $a_j=\{0,1\}$ in a binary alphabet. Zero-vector is k-dimensional tuple, all coordinates of which are equal to zero: $a_j=0,\ j=\overline{1,k}$.

Metrics β of cybernetic (binary) space is defined by a single equality that forms zero-vector for xor-sum of the distances d_i between nonzero and finite quantity of points, closed in a cycle:

$$\beta=\bigoplus_{i=1}^{n} d_i=0.$$

The Hamming distance between two objects (vectors) a and b is determined as derived vector: $d_i=d(a,b)=a_j\bigoplus_{j=1}^{k} b_j$. Otherwise: the metrics β of a vector logic binary space is xor-sum of the distances (it is equal to zero) between finite quantity of graph points (nodes), closed in a cycle. The sum of n-dimensional binary vectors, specifying the coordinates of cycle points, is equal to zero-vector. This metrics definition uses relations that allow reducing the axiom system from three up to one and extending it on any constructions of n-dimensional cyberspace. The classical metrics definition for determining interaction of one, two and three points in vector logic space is a particular case of β-metrics when $i=1,2,3$ respectively:

$$M=\begin{cases} d_1=0\leftrightarrow a=b; \\ d_1\oplus d_2=0\leftrightarrow d(a,b)=d(b,a); \\ d_1\oplus d_2\oplus d_3=0\leftrightarrow d(a,b)\oplus d(b,c)=d(a,c). \end{cases}$$

The metrics β of cybernetic multiple-valued space, where each coordinate of vector (object) is determined in the alphabet that is the Boolean on universe of primitives by the power p: $a_j=\{\alpha_1,\alpha_2,...,\alpha_r,...,\alpha_m\},\ m=2^p$, is the symmetric difference (it is equal to $\varnothing$-vector by all coordinates) of the distances between finite quantity of points, closed in a cycle:

$$\beta=\mathop{\Delta}_{i=1}^{n} d_i=\varnothing . \qquad (1)$$

Equality empty vector the symmetric difference of coordinatewise set-theory interaction (1) emphasizes the equivalence of the components (distances), which form the equation with a single coordinate operation $d_{i,j}\Delta d_{i+1,j}$, used, for instance, in four-digit Cantor's model. It is defined by the corresponding Δ-table:

Δ	0	1	x	$\varnothing$
0	$\varnothing$	x	1	0
1	x	$\varnothing$	0	1
x	1	0	$\varnothing$	x
$\varnothing$	0	1	x	$\varnothing$

$\cap$	0	1	x	$\varnothing$
0	0	$\varnothing$	0	$\varnothing$
1	$\varnothing$	1	1	$\varnothing$
x	0	1	x	$\varnothing$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

$\cup$	0	1	x	$\varnothing$
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
$\varnothing$	0	1	x	$\varnothing$

a	0	1	x	$\varnothing$
$\tilde{a}$	1	0	$\varnothing$	x

(2)

The truth tables for other basic set-theory operations are represented in (2). A number of primitive symbols, formed closed alphabet relative to the set-theory coordinate operations, can be increased. At that the power of alphabet (Boolean) is determined by the expression $m=2^p$, where p is a number of primitive symbols. This metrics is not only of theoretical interest, but has a practical focus on generalization and classification of technical diagnosis problems by creating a model for xor-relations on the set of four main components. The procedures of test synthesis, fault simulation and detection can be reduced to xor-relations on a full interaction graph (Fig. 1) for four nodes (functionality, unit, test, faults) $G=\{F,U,T,L\}$.

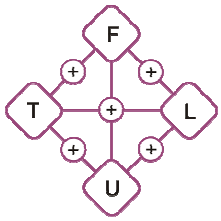


Fig. 1. Graph of interaction between technical diagnosis components

The graph creates four basic triangles, which form 12 triads of relations for the problems of technical diagnosis:

$T\oplus F\oplus L=0$	$T\oplus L\oplus U=0$	$T\oplus F\oplus U=0$	$F\oplus L\oplus U=0$
1) $T=F\oplus L$	4) $T=L\oplus U$	7) $T=F\oplus U$	10) $F=L\oplus U$
2) $F=T\oplus L$	5) $L=T\oplus U$	8) $F=T\oplus U$	11) $L=F\oplus U$
3) $L=T\oplus F$	6) $U=T\oplus L$	9) $U=T\oplus F$	12) $U=F\oplus L$

Insertion of the node U in the graph of interaction between technical diagnosis components extends the functionality of the model; new properties of the resulting system appear. Introduction the new node in the structure has to have strong arguments of its advisability. Concerning the graph, represented in Fig. 1, all problems can be classified into groups as follows.

Group 1 involves the theoretical experiments (on the functionality model), without the device: 1) test synthesis by using the functionality model for a specified fault list; 2) development of the functionality model, based on a given test and fault list; 3) fault simulation for functionality by using given test.

Group 2 – real experiments (by device) without functionality model: 4) test synthesis by physical fault simulation in the device; 5) fault list generation for the device by means of diagnostic experiment; 6) test and faults verification by means of the experiment on a real device.

Group 3 – test experiments (verification) without faults: 7) test synthesis by means of comparing the model simulation results and real device; 8) functionality synthesis by using a real device and a given test; 9) verification of test and functionality model by using the real device with existing faults.

Group 4 – experiments during operation with real inputs: 10) check of correct behavior of a real device on the existing or specified faults; 11) test the device on the existing model in the operation; 12) verification of the functionality and fault list relative to the behavior of a real device.

The most popular problems of the above list are: 1, 3, 5, 8, 9. Another classification of the problem types can be introduced. It allows defining by the graph $G=(F,U,T,L)$ all the conceptual solutions of target problems: test synthesis, functionality model definition, fault model generation and designing of a device:

1) $T=F\oplus L$; 4) $F=T\oplus L$; 7) $L=T\oplus F$; 10) $U=T\oplus L$;
2) $T=U\oplus L$; 5) $F=U\oplus L$; 8) $L=T\oplus U$; 11) $U=T\oplus F$;
3) $T=F\oplus U$; 6) $F=T\oplus U$; 9) $L=F\oplus U$; 12) $U=F\oplus L$.

All constructions, used in a relationship, have the remarkable property of reversibility. Component, calculated using the other two, can be used as an argument to determine any of the two original ones. Thereby, transitive reversibility of each relation triad on complete graph is occur, when by using any two components it is always possible to restore or to determine the third one. At that the format for each component must be identical in structure and dimension (vectors, matrices). Fault diagnosis methods, based on the proposed metrics and testing models, are considered in more detail below.

III. Model for Detecting Functional Failures in Software

The space equation $f(F,T,L,U)=0\rightarrow F\oplus T\oplus L\oplus U=0$ is used. It is transformed to the form $L=(T\oplus F)\oplus(T\oplus U)$. Fault (functional failures) diagnosis is reduced to comparison of simulation $(T\oplus F)$ and full-scale $(T\oplus U)$ results, which generates a functional failure list L, detected in the diagnosed unit. Model-formula for searching the functionally faulty block F_i is reduced to solving by determining xor-interaction between three components:

$$L=F_i\leftarrow[(T\oplus F_i)\overset{p}{\underset{i=1}{\oplus}}(T\oplus U_i)]=0.$$

An analytic model for verification of HDL-code by using temporal assertion engine (additional observation lines) is focused to achievement the specified diagnosis depth and presented as follows:

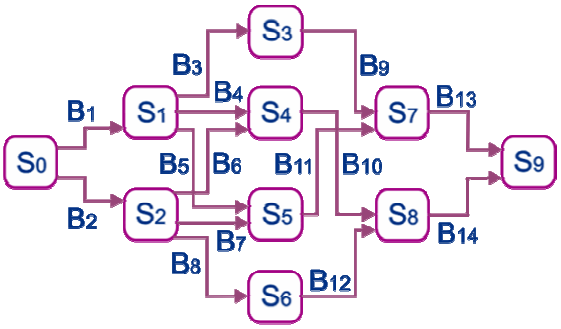
$$\begin{array}{ll} M=f(F,A,B,S,T,L), & F=(A*B)\times S;\ S=f(T,B);\\ A=\{A_1,A_2,...,A_i,...,A_n\}; & B=\{B_1,B_2,...,B_i,...,B_n\};\\ S=\{S_1,S_2,...,S_i,...,S_m\}; & S_i=\{S_{i1},S_{i2},...,S_{ij},...,S_{ip}\};\\ T=\{T_1,T_2,...,T_i,...,T_k\}; & L=\{L_1,L_2,...,L_i,...,L_n\}. \end{array} \quad (3)$$

Here $F=(A*B)\times S$ is functionality, represented by Code-Flow Transaction Graph – CFTG (Fig. 2); $S=\{S_1,S_2,...,S_i,...,S_m\}$ are nodes or states of software when simulating test segments. Otherwise the graph can be considered as ABC-graph – Assertion Based Coverage Graph. Each state $S_i=\{S_{i1},S_{i2},...,S_{ij},...,S_{ip}\}$ is determined by the values of design essential variables (Boolean, register variables, memory). The oriented graph arcs are represented by a set of software blocks

$$B=(B_1,B_2,...,B_i,...,B_n),\ \overset{n}{\underset{i=1}{\cup}}B_i=B;\ \overset{n}{\underset{i=1}{\cap}}B_i=\varnothing,$$

where the assertion $A_i\in A=\{A_1,A_2,...,A_i,...,A_n\}$ can be put in correspondence to each of them. Each arc B_i – a

sequence of code statements – determines the state of the node $S_i = f(T, B_i)$ depending on the test $T = \{T_1, T_2, ..., T_i, ..., T_k\}$. The assertion monitor, uniting the assertions of node incoming arcs $A(S_i) = A_{i1} \vee A_{i2} \vee ... \vee A_{ij} \vee ... \vee A_{in}$ can be put in correspondence to each node. A node can have more than one incoming (outcoming) arc. A set of functionally faulty blocks is represented by the list $L = \{L_1, L_2, ..., L_i, ..., L_n\}$.



$$B = (B_1B_3B_9 \vee (B_2B_7 \vee B_1B_5)B_{11})B_{13} \vee$$
$$\vee ((B_1B_4 \vee B_2B_6)B_{10} \vee B_2B_8B_{12})B_{14} =$$
$$= B_1B_3B_9B_{13} \vee B_2B_7B_{11}B_{13} \vee B_1B_5B_{11}B_{13} \vee$$
$$\vee B_1B_4B_{10}B_{14} \vee B_2B_6B_{10}B_{14} \vee B_2B_8B_{12}B_{14}.$$

Fig. 2. Example of ABC-graph for HDL-code

The model for HDL-code, represented in the form of ABC-graph, describes not only software structure, but test slices of the functional coverage, generated by using software blocks, incoming to the given node. The last one defines the relation between achieved on the test variable space and potential one, which forms the functional coverage as the power of state i-th graph node $Q = \mathrm{card}C_i^r / \mathrm{card}C_i^p$. In the aggregate all nodes have to be full coverage of the state space of software variables, which determines the test quality, equal to 1 (100%): $Q = \mathrm{card}\bigcup_{i=1}^{m} C_i^r / \mathrm{card}\bigcup_{i=1}^{m} C_i^p = 1$. Furthermore, the assertion engine $< A, C >$ that exists in the graph allows monitoring arcs (code-coverage) $A = \{A_1, A_2, ..., A_i, ..., A_n\}$ and nodes (functional coverage) $C = \{C_1, C_2, ..., C_i, ..., C_m\}$. The assertions on arcs are designed for diagnosis of the functional failures in software blocks. The assertions on graph nodes carry information about the quality of test (assertion) for their improvement or complement. The Code-Flow Transaction Graph makes possible the following: 1) use the testability design to estimate the software quality; 2) estimate the costs for creating tests, diagnosing and correcting the functional failures; 3) optimize test synthesis by means of solving the coverage problem by the minimum set of activated paths of all arcs (nodes). For instance, the minimum test for the above mentioned ABC-graph has six segments, which activate all existent paths:

$$T = S_0S_1S_3S_7S_9 \vee S_0S_1S_4S_8S_9 \vee S_0S_1S_5S_7S_9 \vee$$
$$\vee S_0S_2S_4S_8S_9 \vee S_0S_2S_5S_7S_9 \vee S_0S_2S_6S_8S_9.$$

Tests can be associated with the following program block activization matrix:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}
T_1	1	.	1	.	.	.	.	.	1	.	.	.	1	.
T_2	1	.	.	1	.	.	.	.	.	1	.	.	.	1
T_3	1	.	.	.	1	.	.	.	.	.	1	.	1	.
T_4	.	1	.	.	.	1	.	.	.	1	.	.	.	1
T_5	.	1	.	.	.	.	1	.	.	.	1	.	1	.
T_6	.	1	.	.	.	.	.	1	.	.	.	1	.	1

The activization matrix shows the fact of indistinguishability of the functional failures on a test in the blocks 3 and 9, 8 and 12, which constitute two equivalence classes if there is one assertion (monitor) in the node 9. To resolve this indistinguishability it is necessary to create two additional monitors in the nodes 3 and 6. As a result, three assertions in the nodes $A = (A_3, A_6, A_9)$ allow distinguishing all the blocks of software code. Thus, the graph enables not only to synthesize the optimal test, but also to determine the minimum number of assertion monitors in the nodes to search faulty blocks with a given diagnosis depth.

Increasing the number of assertion monitors leads to modification of an activization table. Otherwise, on a given test and the assertion engine it is necessary to solve uniquely the diagnosis problem for functional failures of the software code with the depth up to a software module. At that the number of assertions and test segments to be minimum acceptable for the code identification of all the blocks:

$$|T| + |A| \geq \log_2 |B| = \mathrm{card}T + \mathrm{card}A \geq \log_2 \mathrm{card}B.$$

Initially, the number of monitors-assertions is equal to the number of test segments. The activization table for software modules makes it possible to identify code blocks with functional failures by the generalized output response vector (assertion monitoring)

$$V = (V_1, V_2, ..., V_i, ..., V_n), V_i = \{0,1\}, V_i = T_i \oplus B_j, \forall j(B_{ij} = 1)$$

The vector coordinate $V_i = T_i \oplus B_j = 1$ identifies the nonpassage of the test segment on a subset of activated modules. In accordance with the vector V, defined on the activization table subject to the above rule for calculating its coordinates:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	V
T_1	1	.	1	.	.	.	.	.	1	.	.	.	1	.	0
T_2	1	.	.	1	.	.	.	.	.	1	.	.	.	1	1
T_3	1	.	.	.	1	.	.	.	.	.	1	.	1	.	0
T_4	.	1	.	.	.	1	.	.	.	1	.	.	.	1	1
T_5	.	1	.	.	.	.	1	.	.	.	1	.	1	.	0
T_6	.	1	.	.	.	.	.	1	.	.	.	1	.	1	1

a logical function of software functional failures can be constructed, which is simplified using the coordinates of the output response vector V:

$$B=(\overline{T}_1 \vee B_1 \vee B_3 \vee B_9 \vee B_{13}) \wedge (\overline{T}_2 \vee B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (\overline{T}_3 \vee B_1 \vee B_5 \vee B_{11} \vee B_{13}) \wedge (\overline{T}_4 \vee B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (\overline{T}_5 \vee B_2 \vee B_7 \vee B_{11} \vee B_{13}) \wedge (\overline{T}_6 \vee B_2 \vee B_8 \vee B_{12} \vee B_{14});$$
$$\{V,T\}=(010101) \rightarrow$$
$$B=(0 \vee B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge (0 \vee B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (0 \vee B_2 \vee B_8 \vee B_{12} \vee B_{14}) =$$
$$=(B_1 \vee B_4 \vee B_{10} \vee B_{14}) \wedge (B_2 \vee B_6 \vee B_{10} \vee B_{14}) \wedge$$
$$\wedge (B_2 \vee B_8 \vee B_{12} \vee B_{14}) =$$
$$=B_1B_2 \vee B_4B_2 \vee ... \vee B_3B_6B_{12} \vee ... \vee B_{14}.$$

After transformation the conjunctive normal form (CNF) to disjunctive normal form the obtained terms include all possible solutions in the form of unit coordinate coverage for the output response vector by single or multiple software functional failures. Choosing the best solution is made by determining DNF term of the minimum length.

In this example, the optimal solution is a term containing a single block $B = B_{14}$, which covers three units in the output response vector $V = (010101)$. This fact is also evident from comparison of the last two columns of the activation matrix B.

IV. A METHOD FOR VECTOR LOGIC ANALYZING COLUMNS

Methods for detecting the functional failures (FF) in the statement blocks use previously generated functional failure table $B = [B_{ij}]$, where a row is relation between a test segment and subset of activated (on this segment) software blocks $T_i \approx (B_{i1}, B_{i2}, ..., B_{ij}, ..., B_{in})$. A column forms the relation between software block and test segments $B_j \approx (T_{1j}, T_{2j}, ..., T_{ij}, ..., T_{pj})$, which activate it. Otherwise, a column is an assertion vector, detecting the functional failure in corresponding block. On simulation stage the response $m = (m_1, m_2, ..., m_i, ..., m_p)$ of the assertion engine on a test is identified by means of generating each bit $m_i = (A_1 \vee A_2 \vee ... \vee A_i \vee ... \vee A_k),\ A_i = \{0,1\}$ as response of assertions on the test segment T_i. Searching FF's is based on the definition of xor-operation between the vector of assertion states and columns of the functional failure table $m \oplus (B_1 \vee B_2 \vee ... \vee B_j \vee ... \vee B_n)$. The solution is determined by the vector B_j with minimum quantity of 1 coordinates, which determine the functionally faulty software blocks, checked by the test segments. Diagnosis by the functional failure table on the basis of the response $m = (m_1, m_2, ..., m_i, ..., m_n), m_i = \{0,1\}$ is reduced to the methods for vector logic analyzing columns or rows.

The first one is based on use vector xor-operation between m-response of the functionality on the test, formally considered as an input vector-column, and columns of the fault detection table $m \oplus (B_1 \vee B_2 \vee ... \vee B_j \vee ... \vee B_m)$. To determine the interaction quality of vectors $Q_j(m \oplus B_j)$ and to choose the best solution the columns with minimal quantity of 1's for resultant vector are identified. They forms the functionally faulty blocks, checked by test patterns. The analytic model for solving the diagnosis problem and obtaining the list of functionally faulty software blocks is represented in the following form:

$$L = L \underset{j=1}{\overset{n}{\vee}} B_j \leftarrow \sum_{i=1}^{k} (B_{ij} \overset{k}{\underset{i=1}{\oplus}} m_i) = (0 \vee \min). \quad (4)$$

Here an output response vector is input one for subsequent analyzing of the functional failure table

$$m = f(A,B) \oplus f^*(A,B,L)\ . \quad (5)$$

And it is a result of test experiment – comparison of the functional (output states) for model under test $f(A,B)$ and unit under test $f^*(A,B,L)$ with the faults L on the test patterns A. In second case if a set of faults $L > 1$, it means existence of equivalent functional failures on given test and assertion engine.

A process model for searching the best solution with minimum quantity of 1 coordinates from 2 or more alternatives is shown in Fig. 3. It involves the following operations: 1) Initially, in all coordinates (the worst solution) of the vector Q, where the best solution is stored, 1 values are entered; and simultaneously left slc operation with compaction of 1's is performed for given vector Q_i. 2) Comparing of two vectors is performed: Q and the next estimation Q_i from the solution list. 3) Vector operation And $(Q \wedge Q_i)$ is performed. The result is compared with vector Q, which allows changing it, if the vector Q_i has less quantity of 1 values. 4) The procedure for searching the best solution is repeated by n times.

$$Q = Q\overline{(\vee((Q \wedge Q_i) \oplus Q))} \vee Q_i(\vee((Q \wedge Q_i) \oplus Q));$$
$$Y = \vee((Q \wedge Q_i) \oplus Q);$$
$$Q = Q\overline{Y} \vee Q_i Y.$$

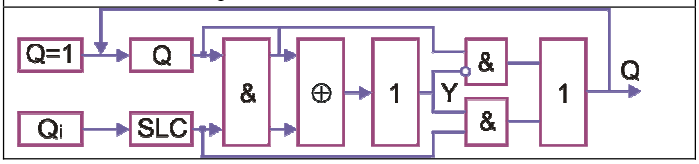
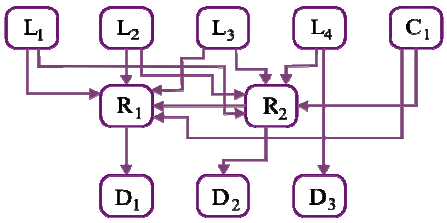


Fig. 3. Process-decision model

An advantage of the method for vector logic analyzing columns is the choice of the best solution from all possible single and multiple faults. Actually, such single functional failures are included in the fault list, which when logical multiplying them by output response vector give a result in the form of vector-column. Disjunction of all columns, generating a solution, is equal to the output response vector

$$\vee_{j=1}^{r} (B_j \in B) = m.$$

An example for analyzing the functional failure table FFT of the module Row_buffer (Fig. 4) is represented below.



Test	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	m_1	m_2
	L_1	L_2	L_3	L_4	C_1	R_1	R_2	D_1	D_2	D_3		
T_1	1	0	0	0	0	1	0	1	0	1	0	1
T_2	0	1	0	0	0	0	1	0	1	0	1	1
T_3	0	0	1	0	0	1	1	1	0	1	0	1
T_4	0	0	0	1	0	0	1	0	1	0	1	0
T_5	0	0	0	0	1	1	1	1	0	1	0	0
T_6	1	0	0	0	0	0	1	0	1	0	1	1
T_7	0	1	0	0	0	1	0	1	0	0	0	1
T_8	0	0	1	0	0	1	0	1	0	0	0	1
T_9	0	0	1	0	0	0	1	0	1	0	1	0
T_{10}	0	0	0	1	0	1	1	1	0	0	0	0
T_{11}	0	0	0	1	0	0	0	0	0	1	0	0
T_{12}	0	0	0	0	1	0	1	0	1	0	1	0
T_{13}	0	0	0	0	1	1	1	1	0	0	0	0

Fig. 4. Row_buffer transaction graph and table FFT

On the basis of the diagnosis procedure (4) and tables FFT (see Fig. 3) the faulty components can be determined by analysis of FFT columns. Here the vectors m_1, m_2 define the diagnosis results, performed by the procedure (5). The diagnosis result for single and multiple functional failures is following:

$$L^s(m_1) = m_1 \wedge (\vee_{j=1}^{10} B_j) = B_9 \to D_2;$$
$$L^m(m_2) = m_2 \wedge (\vee_{j=1}^{10} B_j) = B_1 \vee B_2 \to L_1 \vee L_2;$$
$$Q(m_1, D_2) = 1;\ Q[m_2, (L_1 \vee L_2)] = \frac{1}{3}(\frac{4}{13} + \frac{1}{4} + 1) = 0{,}52.$$

In the first case, the diagnosis is defined as a single faulty module D_2 that present in the transactional graph; the solution quality is equal to 1. In the second case, the diagnosis procedure detects two faulty modules $L_1 \vee L_2$, the quality estimation of which is not the optimal. Nevertheless, the solution is the best among all the possible, which is maximally approximate to the output response vector by the membership criterion $Q[m_2, (L_1 \vee L_2)]$. The computational complexity of the method for analyzing columns is determined by the following dependence: $Z^c = 3n^2 + n^2 = 4n^2$; $Z^r = 3n + n = 4n$. Here, the first estimate takes into account the implementation of coordinate operations on the matrix of the dimension $n \times n$. The second estimate determines the computational complexity of the register parallel operations to compute quality criteria and process the matrix, respectively.

V. Method for Vector Logic Analysis of Rows

The method is designed for determination of fault or functional failure (FF) location in software code and consists of two procedures: 1) determining the logical product of the conjunction of lines, marked by unit values of the vector $T_i(m_i = 1)$, by the negation of disjunction of zero rows $T_i(m_i = 0)$ for single faulty modules; 2) determining the logical product of disjunction of unit lines by the negation of the disjunction of zero rows for multiple faulty modules:

$$L^s = (\wedge_{\forall m_i = 1} T_i) \wedge (\overline{\vee_{\forall m_i = 0} T_i});$$
$$L^m = (\vee_{\forall m_i = 1} T_i) \wedge (\overline{\vee_{\forall m_i = 0} T_i}); \qquad (6)$$

The formulas are interesting, because they are not related to the diagnosis quality criteria and operate only two components: FFT table and output response vector. Performing the diagnosis procedure by the formulae (4) for the output response vector $m_1 = (0101010010010)$, specified in the last table FFT, forms the result: $L^s(m_1, T) = D_2$, which is not worse than previously obtained by the method for analyzing columns. For the output response vector $m_2 = (1110011100000)$ the diagnosis result is: $L^m(m_2, T) = L_1 \vee L_2$. Computational complexity of the method for analyzing rows is determined

by the following dependence: $Z^c = n^2$; $Z^r = n$. The first estimate is designed to count the number of coordinate operations, the second one determines the computational complexity of processing, based on the register parallel operations. The proposed methods for diagnosing functional failures in software and hardware are the most important components of the Infrastructure IP.

Formulae (6) can be modified if the following designations are introduced:

$$a = (\underset{\forall m_i=1}{\wedge} T_i);\ b = (\underset{\forall m_i=0}{\vee} T_i);\ c = (\underset{\forall m_i=1}{\vee} T_i);$$

$$L^s = a\bar{b} = a \oplus ab = a(a \oplus b) = a(b \oplus 1);$$

$$L^m = c\bar{b} = c \oplus cb = c(c \oplus b) = c(b \oplus 1);$$

$$L = \begin{cases} a\bar{b} = a\bar{b} = a \oplus ab = a(a \oplus b) = a(b \oplus 1); \\ c\bar{b} = c\bar{b} = c \oplus cb = c(c \oplus b) = c(b \oplus 1) \leftarrow a\bar{b} = 0 \end{cases}$$

Any right side expression of the equations can be used to detect functional failure in the software or hardware. The difference lies in the presence or absence of inversion, which is replaced by xor-operation, more preferable for diagnosis and pattern recognition. In this case, the process model for diagnosing single (using a-component) or multiple (b-component) faults (functional failures) based on analyzing the table FFT has an effective vector-oriented computing technology:

$$L = (b \oplus 1)(a \vee c),$$

embedded Infrastructure IP of software/hardware. According to set theory, this means determining the result of set-theory subtraction $L = (a \vee c) \setminus b = (a \setminus b) \vee (c \setminus b)$ in the algebra-logic vector space. For such operations the multimatrix processor is needed, which is strictly focused on the parallel execution of several logic operations on data matrices.

VI. Matrix method for detecting the functional failures in software

Further to the software transaction graph (3) a method for diagnosing functional failures in software uses the triad of matrices of the same format:

$$M = B \oplus A \oplus L = 0,$$

$$L = B \oplus A \leftarrow L_{ij} = B_{ij} \oplus A_{ij} \leftarrow \{B_{ij}, A_{ij}, L_{ij}\} = \{0,1\};$$

$$B = [B_{ij}],\ A = [A_{ij}],\ L = [L_{ij}],$$

$$i = \overline{1,n};\ j = \overline{1,m};\ \oplus = a\bar{b} \vee \bar{a}b.$$

Here matrices form: B – block activization on test segments during simulation; A – activity of assertions, corresponding to blocks, on test segments and during simulation; L – faulty blocks, obtained as result of xor-operation on two above matrices. Coordinate-wise analyzing the matrices uses binary xor-operation, such as (see Table I).

Obtained result $L = B \oplus A$ in the form of L-matrix $[L_{ij}] = (T \times B \times \{0,1\})$, all coordinated of which are equal to zero, indicates absence functional failures in software relatively the proposed verification plan in the format (test – functional blocks – activization $[B_{ij}] = (T \times B \times \{0,1\})$, test – assertions – response $[A_{ij}] = (T \times A \times \{0,1\})$. Another model experiment indicates presence the functional failures $L = \{B_1, B_2, B_3, B_5, B_6\}$ in software code (see Table II).

Here are the results of vector operations on all rows of two tables $\vee L_i$=11101100 and $\vee A_i$=11011111. Logical conjunction of them with the preliminary inversion of the first vector gives the coordinates of blocks with functional failures, marked by units. In this example, the vector forms only one block (00100000)&(11101100)=(00100000).

Tables I, II

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	.	.	.	.	.	.	.	.
T_2	.	.	.	.	.	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	.	.	.	.	.	.	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	.	.	.	.	.	.

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	1	.	.	1	.	.	1	.
T_2	.	0	0	.	0	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	0	.	0	.	.	0	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	0	1	1	.	.	.
$\vee A_i$	1	1	0	1	1	1	1	1

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.
$\vee L_i$	1	1	1	0	1	1	0	0

What is the reason for the reduction of faulty blocks? If to assume that in compliance with the verification plan the verification of the first block has to detect faults on first and sixth test, which is not satisfied, so block 1 can be excluded from the fault list. Similarly, modules 2, 5, 6 can be excluded. Then the corrected result will have only one block with the functional failures: $L = \{B_3\}$. The procedure for refining the diagnosis result can also be formalized in the following form: $L = B_j \leftarrow B_j \wedge L_j = B_j, j = \overline{1,m}$. If the comparison result is negative $B_j \oplus L_j = 0$, it means the code is incorrect, assertion or test failed, including functional coverage. For the diagnosis code in accordance with the process model of the form

$$L(B,T) = (B \oplus A) \rightarrow L(B) = (\overline{\underset{i=1,n}{\vee} A_i}) \wedge (\underset{i=1,n}{\vee} L_i),$$

it is necessary to consider the following items:

1. Coverage is any metric for choosing test and determining its confidence. Code coverage is test metric, focused on the confirmation of execution of all code lines. Decomposition of software code into blocks is performed $B = \{B^s, B^t\} \leftarrow B^s \cap B^t = \varnothing, B^s \cup B^t = B$. Each block belongs to one of two types: the sequence of statements without a branch or time delay circuit $B_i \in \{B^s, B^t\}$. Location of assertion monitors is carried out for block activity on test at the beginning of the branch or in the first timer cycle of a time delay circuit. In the modeling process assertions form an activization matrix for software blocks on each test segment $B_{ij} = T_i \oplus B_j \in \{0,1\}$. If the block is active (assertion passed) on the test (testbench), matrix coordinate is equal to 1, otherwise – $B_{ij} = 0$. Testbench is input conditions for testing the HDL-code and corresponding output responses, which define transformations of the device under test in the functional subspace.

2. Functional coverage is test metric that ensures the accessability of all essential states in the software variable and function definition space. Decomposition of software functionality in control and transaction graphs is performed: $F = \{F^c, F^t\} \leftarrow F^c \cap F^t = \varnothing, F^c \cup F^t = F$. This makes it possible to considerably reduce the dimension of coverage problem that defines the domain for the control variable and data flow. Test generation and the subsequent coverage driven verification use the above mentioned graphs with constraints, taken from the specification. Synthesized test for the control graph allows activation of all logic and arithmetic variables involved in initiation of software transaction. Way of variable activation or test synthesis consists of pseudo-random or deterministic (algorithmic) generating test inputs, as well as hand-writing input stimuli. Forms of coverage definition are an abbreviated truth table, Boolean equation, binary decision diagrams, the flowgraph. Test for the second graph handles data flows, which at the system level not always have to be checked because of the absence of faults, such as short circuits between the variables or constant faults in them. Transaction graph can be used to create a verification plan for essential interface parameters of software. To do this it is necessary to use interface assertions operating by global variables.

2. Assertion matrix for software blocks has a form similar to the structure of block activation $A = [A_{ij}]$. Here format of assertion as logic statement, using the essential variables of software block $f(X) = A_{ij} = \{0,1\}$, responses for running the corresponding activated on the test module $B_{ij} = 1$. Several statements can be in the block, separated to increase the diagnosis depth or united by function or. In last case assertion responses for correct functioning of the block. Assertion has two values: 1 – block operates fault-free, 0 – there are functional failures. Assertions are represented by two hierarchy levels: interface and block ones $A = \{A^i, A^b\}$. The first ones are focused on testing the essential parameters of the specifications, which are common for the software and external for it. Second ones are built into software block, which don't have branches. Power of commands or code lines – up to 20 – is determined by the number of statements to be placed on the screen. Such block can contain time or event delay statements.

VII. Implementation of Models and Methods in the Verification System

Practical implementation of models and verification methods is integrated into the simulation environment Riviera of Aldec Inc., Fig. 5. New assertion and diagnosis modules, added in the system, improved the existing verification process, which allowed 15% reduction the design time of digital product.

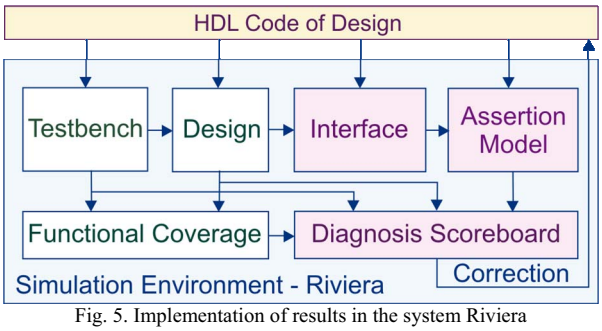


Fig. 5. Implementation of results in the system Riviera

Actually, application of assertions makes possible to decrease the length of test-bench code and considerably reduce (x3) the design time (Fig. 6), which is the most expensive. Assertion engine allows increasing the diagnosis

depth of functional failures in software blocks up to level 10-20 HDL-code statements.

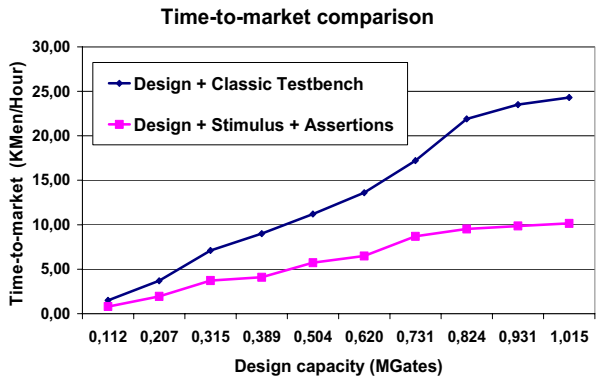


Fig. 6. Comparative analysis of verification methods

Due to the interaction of simulation tools and assertion engine, automatically placed inside the HDL-code, an access of diagnosis tools to the values of all internal signals is appeared. This allows quickly identifying the location and type of the functional failure, as well as reducing the time of error detection in the evolution of product with top-down design. Application of assertion for 50 real-life designs (from 5 thousand up to 5 million gates) allowed obtaining hundreds of dedicated solutions, included in the verification template library VTL, which generalizes the most popular on the market EDA (Electronic Design Automation) temporal verification limitations for the broad class of digital products. Software implementation of the proposed system for analyzing assertions and diagnosing HDL-code is part of a multifunctional integrated environment Aldec Riviera for simulation and verification of HDL-models.

High performance and technological combination of assertion analysis system and HDL-simulator of Aldec company is largely achieved through integration with the internal simulator components, including HDL-language compilers. Processing the results of the assertion analysis system is provided by a set of visual tools of Riviera environment to facilitate the diagnosis and removal of functional failures. The assertion analysis model can also be implemented in hardware with certain constraints on a subset of the supported language structures. Products Riviera including the components of assertion temporal verification, which allow improving the design quality for 3-5%, currently, occupies a leading position in the world IT market with the number of installations of 5,000 a year in 200 companies and universities in more than 20 countries on the world.

VIII. Multimatrix processor of binary operations and verification infrastructure

To implement effective computational processes by time and cost of associated with the diagnosis of functional failures it is necessary processor of the simple architecture with minimum instruction set, where the operands are not only Boolean variables, but also more complex structures such as registers and matrices. Such processor should execute in parallel mode operations over all bits of the regular operands, not requiring special compilers for paralleling computing processes.

Multimatrix processor (MMP) is a minimum architecture of instruction primitives, where each of them focused on the parallel execution of only one operation (and, or, xor, slc) over the corresponding matrix (two-dimensional data array). The number of command-oriented matrix primitives creates a system – a heterogeneous multimatrix processor of binary operations with buffer M, Fig. 7.

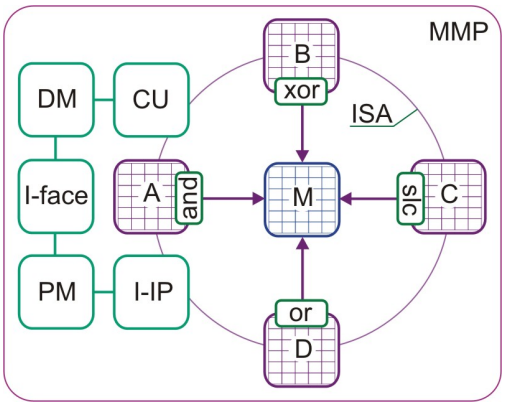


Fig. 7. Multimatrix processor of binary operations

The standard blocks are shown here: data DM and program PM memory, control unit CU, interface I-face and infrastructure I-IP, as well as multimatrix processor, including 4 memory blocks with embedded operations (A – and, B – xor, C – or, D – slc – shift left crowding) and buffer memory M. Multimatrix processor (MMP) is focused on parallel execution one of four instructions (ISA – Instruction Set Architecture) for processing matrices of binary data of the same dimension $M = M\{\text{and}, \text{or}, \text{xor}, \text{slc}\}\{A, B, C, D\}$ and saving the result in the buffer M. Feature of MMP is that each instruction has data matrix for parallel processing (not matrix cell has instruction set of 4 operations) to simplify the control structure and device in whole. The complexity of MMP is focused on data structures, matrix memory has a single hardware embedded instruction that enables to implement primitive control system for parallel computing (SIMD – Single Instruction Multiple Data). Proposed MMP architecture is adapted to execution of logic instructions by the operands of register level. MMP prototype is integrated in the hardware acceleration board for simulation and verification HES™, Aldec Inc.

On the basis of multimatrix (register) processor an infrastructure for verification HDL-code (Fig. 8) is developed. It is modification of I-IP standard IEEE 1500 SECT [3, 4, 11, 14]. There are 4 process models: testing on the simulation stage, diagnosis of functional failures, diagnosis optimization, repairing.

1. Process model for testing involves HDL-model, assertion engine, testbench and coverage. Last one estimates test quality for all design states. In simulating the

activization matrix B for software blocks and assertion response matrix A on test segments are generated. Matrix A can be transformed to assertion state vector m by application of the function Or to vector-columns of A-matrix.

$$\begin{cases} B = (T \oplus F); \\ m = \bigvee_{j=1}^{m} A_j \leftarrow A = (T \oplus A^c). \end{cases}$$

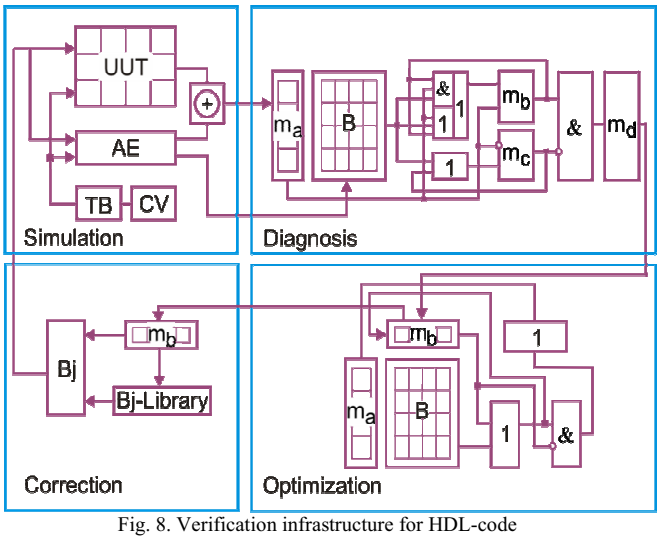


Fig. 8. Verification infrastructure for HDL-code

2. The last two components are used in the second process model for diagnosing blocks of HDL-code. Diagnosis is fault vector, which forms a subset of blocks m_d with functional failures. At that the errors can be in testbench and in assertion statements, which are designed for testing and monitoring software blocks. If exact identification of the block is absent when comparing the columns of activization matrix and assertion responses, triple diagnosis uncertainty $D = \{B_j, T_i, A_{ij}\}$ arises.

3. The third block solves the problem of minimizing the number of blocks, in which functional failures can be, up to one of them. At that a block activization matrix and the diagnosis m_d, obtained in the previous process model, are used.

4. Correction of functional failures is focused on manual searching errors in a software block, presented by the vector m_b. Automated correcting errors in the block is possible, if there is a library of diversion software modules of the similar functionality in the verification infrastructure.

The proposed infrastructure is one of steps towards the creation of verification automaton for software blocks. An example of diagnosing the functional failure, based on using the activization matrix, is represented below. The vector of assertion responses is obtained from the matrix $A_{ij} = \{1 \rightarrow \text{failed}, 0 \rightarrow \text{passed}\}$ by disjunctive union of rows content:

$$m_i = \bigvee_{j=1}^{m} A_{ij} =$$

T	m
T_1	.
T_2	1
T_3	.
T_4	1
T_5	.
T_6	.
T_7	.
T_8	1

=

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.

Subsequent implementation of xor-operation between the assertion vector and activization matrix columns allows obtaining the best solution, which is determined by the minimum code distance

$$L = L \vee B_j \leftarrow \sum_{i=1}^{n} (B_{ij} \overset{n}{\oplus}_{i=1} m_i) = (0 \vee \min):$$

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

T	m
T_1	.
T_2	1
T_3	.
T_4	1
T_5	.
T_6	.
T_7	.
T_8	1

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	1	.	.	1	.	1	1	1
T_3	.	.	.	.	.	1	1	1
T_4	.	1	.	1	1	.	1	1
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	1	1	.	.	.	1	1	1
$d(A,B_j)$	4	4	0	4	2	4	6	6

Diagnosis is block 3 has functional failures, because three assertions are failed on the test segments 2,4 and 8, which in this combination activate only block number 3. If assertion matrix (not vector) is used for diagnosing, searching for faulty blocks is the following:

B_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	1	.	1	.	.	1	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	1	1	1	.	.	.

$\oplus$

A_{ij}	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
T_1	.	.	.	.	.	.	.	.
T_2	.	1	1	.	1	.	.	.
T_3	.	.	.	.	.	.	.	.
T_4	1	.	1	.	.	1	.	.
T_5	.	.	.	.	.	.	.	.
T_6	.	.	.	.	.	.	.	.
T_7	.	.	.	.	.	.	.	.
T_8	.	.	1	.	.	.	.	.

=

L_{ij}	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
T_1	1	.	.	1	.	.	1	.
T_2	.	.	.	.	.	.	.	.
T_3	.	.	.	.	.	1	1	1
T_4	.	.	.	.	.	.	.	.
T_5	.	1	.	1	.	.	.	1
T_6	1	.	.	.	.	.	1	1
T_7	.	1	.	.	1	1	.	.
T_8	.	.	.	1	1	.	.	.
$d(A,B_j)$	2	2	0	3	2	2	3	3

Diagnosis is similar to the previous one: block 3 has functional failures, because the code distance is equal to zero only for the column number 3.

IX. Conclusion

The following results are proposed in the paper:

1. A structural model for relations on the set of four main components of technical diagnosis (functionality, unit, test and faults), which is characterized by complete xor-interaction of all the graph nodes and transitive reversibility of each relation triad that allows defining and classifying the ways of solving practical problems, including test synthesis, fault simulation and fault detection.

2. A new model of software in the form of Code-Flow Transaction Graph, as well as a new matrix method for diagnosing functional failures, which are characterized by adaptability of data preparation when detecting faulty

blocks, are proposed. They allow considerably reducing the design time of digital systems on chips.

3. Methods for searching functional failures, which differ in parallel execution of vector operations on the rows of a functional failure table, are improved. They allow substantially (x10) increasing the performance of computational procedures associated with diagnosis and repair of software and hardware.

4. The architecture of multimatrix processor, focused to increasing the speed of embedded diagnosis of functional failures in the software or hardware product, which differs using parallel logic vector operations and, or, xor, slc that enables to increase considerably (x10) the speed of diagnosing single and/or multiple faults (functional failures).

5. The infrastructure for verification and diagnosis of HDL-code for design digital systems-on-chips, which involves four process models for testing, diagnosing, optimization and correcting errors, closed in a cycle, that makes it possible to reduce the time of code debugging, when creating a design.

6. Practical implementation of models and verification methods is integrated into the simulating environment Riviera of Aldec Inc. New assertion and diagnosis modules improved the existing verification process, which allowed 15% reduction in overall design time of digital products.

REFERENCES

[1] Technical diagnosis basics / Editor P.P. Parchomenko.- M.: Energy.– 1976.– 460 p.

[2] Parchomenko P.P., Sogomonyan E.S. Technical diagnosis basics (Optimization of diagnosis algorithms, hardware tools) / Editor P.P. Parchomenko.– M.: Energy.– 1981.– 320 p.

[3] Infrastructure for brain-like computing / M.F. Bondaryenko, O.A. Guz, V.I. Hahanov, Yu.P. Shabanov-Kushnaryenko.– Kharkov: Novoye Slovo.– 2010.– 160 p.

[4] Design and Verification of digital systems on chips / V.I. Hahanov, I.V. Hahanova, E.I. Litvinova, O.A. Guz.– Kharkov: Novoye Slovo. – 2010. – 528 p.

[5] Semenets V.V., Hahanova I.V., Hahanov V.I. Design of digital systems by using VHDL language.– Kharkov: KHNURE.– 2003.– 492 p.

[6] Hahanov V.I., Hahanova I.V. VHDL+Verilog = synthesis for minutes. – Kharkov: KHNURE.– 2006.– 264 p.

[7] Hahanov V.I. Technical diagnosis of digital and microprocessor structures: Manual.- K.: ISIO, 1995.- 242 p.

[8] Skobtsov Yu.A. Logic simulating and testing digital devices / Yu.A. Skobtsov, V.Yu. Skobtsov. –Donetsk: IPMM NSA of Ukraine, DonNTU.– 2005.– 436 p.

[9] IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture IEEE Std 1149.7-2009. – 985 p.

[10] Da Silva F., McLaurin T., Waayers T. The Core Test Wrapper Handbook. Rationale and Application of IEEE Std. 1500™. –Springer.– 2006.– XXIX.– 276 p.

[11] Marinissen E.J., Yervant Zorian. Guest Editors' Introduction: The Status of IEEE Std 1500.–IEEE Design & Test of Computers.– 2009.– No26(1).– P.6-7.

[12] IEEE Std 1800-2009 IEEE Standard for System Verilog-Unified Hardware Design, Specification, and Verification Language. http://ieeexplore.ieee.org/servlet/opac?punumber= 5354133

[13] Marinissen E.J. Testing TSV-based three-dimensional stacked ICs // DATE 2010.– 2010.– P.1689-1694.

[14] Benso A., Di Carlo S., Prinetto P., Zorian Y. IEEE Standard 1500 Compliance Verification for Embedded Cores // IEEE Trans. VLSI Syst.– 2008.– No 16(4).– P. 397-407.

[15] Ubar R., Kostin S., Raik J. Embedded diagnosis in digital systems // 26th International Conference “Microelectronics”, MIEL 2008. – 2008.– P. 421-424.

[16] Elm M., Wunderlich H.-J. Scan Chain Organization for Embedded Diagnosis // Design, Automation and Test in Europe, DATE '08.– 2008.– P. 468–473.

[17] Bulent I. Dervisoglu. A Unified DFT Architecture for Use with IEEE 1149.1 and VSIA/IEEE P1500 Compliant Test Access Controllers. Proceedings of the Design Automation Conference. – 2001. – P. 53-58.

[18] Chenlong Hu, Ping Yang, Ying Xiao, Shaoxiong Zhou. Hardware design and realization of matrix converter based on DSP & CPLD // 3rd International Conference Power Electronics Systems and Applications.– 2009.– P. 1-5.

[19] Dave N., Fleming K., Myron King, Pellauer M., Vijayaraghavan M. Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA // 5th IEEE/ACM International Conference Formal Methods and Models for Codesign.– 2007.– P.97-100.

[20] Loucks W.M., Snelgrove M., Zaky S.G. A Vector Processor Based on One-BitMicroprocessors // IEEE Micro.–Volume 2, Issue 1.– 1982.– P. 53-62.

[21] Hilewitz Y., Lauradoux C., Lee R.B. Bit matrix multiplication in commodity processors // International Conference Application-Specific Systems, Architectures and Processors.– 2008.– P. 7-12.

[22] Soon, J.L.K.; Low Ching Ling; DEV. Design explorer for verification. Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium: 2009, Page(s): 413 – 416.

[23] Rafe, V.; Rafeh, R.; Azizi, S.; Miralvand, M.R.Z.; Verification and Validation of Activity Diagrams Using Graph Transformation. Computer Technology and Development, 2009. ICCTD '09. 2009 , Page(s): 201-205.

[24] Xiaoxi Xu; Cheng-Chew Lim; Using Transfer-Resource Graph for Software-Based Verification of System-on-Chip. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume: 27, Issue: 7. 2008, Page(s): 1315 – 1328.

[25] Zhongjun Du; Zhengjun Dang; A New Algorithm Based Graph-Search for Workflow Verification. Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference: 2010, Page(s): 1–3.

[26] Gorbatov V.A., Gorbatov A.V., Gorbatova M.V. Discrete mathematics.– M: High School, 2006.– 448 p.