

Testability analysis method for hardware and software based on assertion libraries

Maryna Kaminska, Roman Prikhodchenko, Artem Kubirya, Pavel Mocar
Kharkov National University of Radioelectronics
Lenin ave, 14, Kharkov, 61200, Ukraine
Phone: (+380 57) 7011326; e-mail: kaminska_maryna@mail.ru

1. Abstract

Testability analysis method for software and hardware products, which represented by system level as a composition of control and operational automata is offered. Methodology of bottlenecks selection for assertions implementation in program code is proposed.

2. Introduction

Today digital systems on chip are complex systems, which consist of huge number of components. Structural and functional complexity of such systems could increase possibility of defects and faults occurring. So verification is necessary and actual procedure during design process. Actuality is defined by necessity of simulation tools performance and test quality increasing, and test size decreasing for circuits which consist of millions gates. Software, in turn, also could consist of huge number of functionalities, which are needed to be completely verified. High costs, resulting from verification functionally and structurally complex circuits can be to 70% from whole design time. To decrease these costs, testable design standards and Ad-Hoc technologies could be used to decrease testing time and engineers efforts.

Complex system could be presented as hierarchical system. Verification and test procedure should be executed on each level of hierarchy. In test technologies of complex systems such approaches as Design for Manufacturability, Design for Testability, and Design for Verification could be used [1-15].

Here it is necessary to clear what it means testability. Testability it is device property oriented on keeping of time and money expenses in given limit. Testability shows how easy given device could be tested and verified using additional equipment and Ad-Hoc methods on separate design stages. Methods of testable design could be divided in two groups: 1) Structural-functional object analysis, numerical estimation of controllability, observability and testability for circuit. Such analysis could be used on design stage; 2) Ways of structural design of testable and self tested circuits, based on using scan path, which allows to observe internal nodes of circuit.

The low level of device testability leads to increasing of number of non-tested faults and verification time at design, production and operations stages. Therefore, the

cost of diagnostic (a degree of faults concentration) decreases essentially during techniques of testability design.

Using of testable design approaches allows to essentially increasing quality of designed product and ensuring effective testing and verification procedure on the earliest stages of developing at the expense of permissible area and time overheads. Time overheads are defined by using of assertions in program code, but such delay is insignificant in comparison with sufficiently decreasing of defects number. Also assertions accelerate time to market.

By principle of academician Glushkov V.M., each device could be represented as combination of control automata and operational automata. Such approach simplifies design procedure and facilitates understanding of circuit functionality. Operational automata could be presented by system level (before synthesis representation of circuit, its internal structural model), by register transfer level and gate level. For testability analysis of operational automata this one could be presented as graph, which consist of vertices and oriented arcs. Each vertices defined by set of components: input, output variables, register variables, ALU, memories. Arcs presented by composition of operands for information transmission between vertices. Structure of digital device represented on figure 1 as composition of control automata and operational automata.

Proposed automata model represented by two basic components M^{OA}, M^{CA} – operational and control automata. Model on the figure corresponds to following analytic structure:

$$\begin{cases} M = (M^{OA}, M^{CA}); \\ M^{OA} = \{\vec{X}^O, Y^C, Y^O, \vec{Z}^O\}; M^{CA} = \{X^C, Y^O, Y^C, Z^C\}; \\ \vec{Z}^O = f^O(\vec{X}^O, Y^C); Y^O = g^O(\vec{X}^O, Y^C); \\ Z^C = f^C(X^C, Y^O); Y^C = g^C(X^C, Y^O). \end{cases}$$

Here \vec{X}^O, \vec{Z}^O – vector or register input and output variables; Y^C, Y^O, Z^C – control signals, information signals, monitoring signals; $f^O, g^O(f^C, g^C)$ – functions, which define relevance between interface signals in control automata and operational automata.

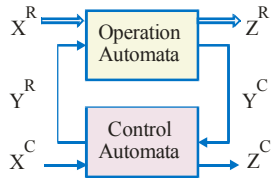


Figure 1. Composition of control and operational automata

In paper structural method of device analysis or program code is offered. Proposed method will allow to sufficiently decreasing time expenses during testing and verification procedure and increase yield ration using of assertions in program code.

Object under analysis is digital device or software product, represented as program code on system level.

Tasks: 1. Create of automata model of program product or module, which consist of operational and control automata; 2. Synthesis of control automata; 3. Synthesis of operational automata [16,17]; 4. Full model represented by register transfer graph; 5. Testability analysis method for control and operational automata and algorithm of circuit (program code) modification and using assertion based on testability analysis; 6. Experimental approbation of proposed method.

3. Design techniques evolution

Developers' possibilities and test procedure efficiency increases on high-level description. New technologies growing based on specifications and new standards development, which allow to open gates on world market becomes popular and important moment in design process. Thus, register level languages progress leads to progress in digital devices synthesis tools. Ensuring of correct device functionality on earliest level of description is important and hard task for developers. Main developer's target is verification time decreasing that requires new effective technologies for testing and verification. Verification could be divided on dynamic verification (fault simulation) and static (formal) verification, for example, using of assertions [18-20].

Formal verification is used for exhaustive verification of model properties and functionality based formal techniques, that allow to avoid most part of defect on the earliest design stages.

Along with assertions different Ad-Hoc technologies could be used. Based on such technologies, additional test logic could be added to device for scan internal bottlenecks in device. Ad-Hoc technologies allow to increase controllability and observability of device (product) functional blocks and lines. Such approach allows decreasing time to market and increasing product quality owing to insignificant area overhead.

In this work method of testability analysis for internal model of program code is offered. Additional tested logic or assertions could be added to device or program product based on obtained characteristics of controllability and testability. Lines and functional blocks with worst values of controllability and observability could be selected as control points for additional verification. Today number of rulers exists for assertions usage, but absent exact method of bottlenecks selection. Today only developer can place assertions in code based on his knowledge

about program bottlenecks and intuition. Testability analysis and bottlenecks selection could be executed automatically without developers and engineers efforts. Therefore, method of testability analysis and algorithm of bottlenecks selection and assertions placement was proposed.

Today assertions become very popular due to possibility use them in any device specification or program code and simplicity of assertions usage.

Definition: Assertion is expression on system level, which could define the transformation correctness during design process relative to description of current stage or specification requirements.

According to definition assertion could be presented as predicate:

$$L_i = f(L_{i1}, L_{i2}, \dots, L_{ij}, \dots, L_{in}) = Y = \{0,1\},$$

which could be equal to «false» or «true» on a set of variables n_i [25].

There are three types of assertions: fault detection, functional paths, and possible states of design. Such division allows to check not only calculation mechanism (operational automata), but control automata also. Control automata could be presented as FSM, and each of states should be checked. [25–28]. Here we could talk about testability analysis for FSM. Test diagnosis mechanism and algorithm of assertions placing are also offered in this work.

4. Method of structural analysis of operational automata

Along with growing of SoC complexity, assertions become more popular on system level where circuit could be represented as program code.

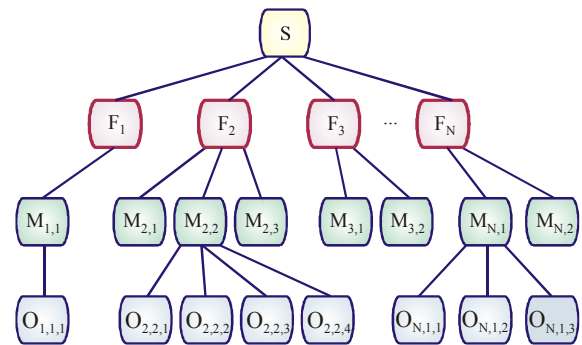


Figure 2. Hierarchical presentation of software (hardware) products

SoC refining on system level could be presented as set of functionalities. Functionality could be represented as set of methods. Such structure is presented on fig. 2. On figure 3 diagnosis architecture based on assertions usage is represented. IEEE 1149.1 BS and IEEE 1500 SECT are used as prototype of proposed test algorithm.

Here F1-F4 are device functionalities. Input registers RI are used for temporary storing of test data, which could be shifted to functionalities by diagnosis controller. Output registers RO contain test results – logic zeros and ones. “Ones” signalize about defect in functional blocks. Output dates from registers could be recorded as n-bit vector $V = \{V_1, V_2, \dots, V_i, \dots, V_n\}$, where n is number of functional blocks in device or program code.

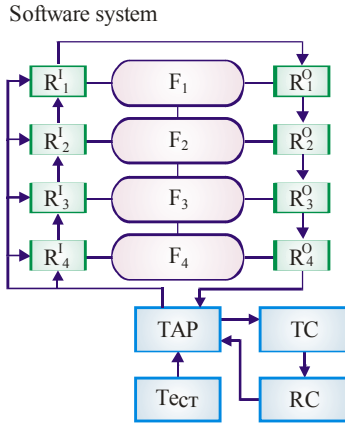


Figure 3. System diagnosis with assertions

For definition of minimal number of test cases algebraical method could be used.

As additional analysis for each functional blocks testability analysis methods could be used. And each functional block could be represented as two graph components with oriented arcs. Such model is proposed instead to model on figure 1, because automata model $M = (M^{OA}, M^{CA})$ is unsuitable for practical tasks of testable design.

Graph model could be represented by following structure:

$$\left\{ \begin{array}{l} M = (M^{OR}, M^{CG}), \\ M^{OR} = \{R, I\}; \\ R = \{R_1, R_2, \dots, R_i, \dots, R_n\}, I = \{I_1, I_2, \dots, I_j, \dots, I_m\}; \\ R_i = f(R_k, I_j); \\ M^{CG} = \{S, T\}; \\ R = \{S_1, S_2, \dots, S_i, \dots, S_p\}, T = \{T_1, T_2, \dots, T_j, \dots, T_q\}; \\ S_i = f(S_k, T_j). \end{array} \right.$$

Here M^{OR} – register transfer graph by S.G. Sharshunov, which consist of set of vertices R , represented by all memory components (registers, triggers, counters, memory, input and output buses) which used in program, and arcs with instructions I , which activate information transferring between vertices. Expression $R_i = f(R_k, I_j)$

– definition of functional dependency between adjacent vertices $R_i \rightarrow R_k$ which joined for execution of operation $I_j \in I$. Component M^{CG} corresponds to substantial graph of control automata, which defined on a set of vertices S , joined by arcs T with conditional transitions. Expression $S_i = f(S_k, T_j)$ defines functional dependency between adjacent vertices $S_i \rightarrow S_k$ of control automata, which joined for transition $T_j \in T$.

Structure of graph represented below:

$$G = \{R, I\}; R = \{R_1, R_2, \dots, R_n\}, I = \{I_1, I_2, \dots, I_m\}; \\ I_{ij} \in I_i \approx (R_p R_q);$$

Here model of software (hardware) module represented by graph $G = \{R, I\}$, which consist of vertices (registers) and arcs (instructions). Each arcs marked by not less than one operation $I_{ij} \in I_i \approx (R_p R_q)$, which forms subset of

instructions, which belongs to arc $(R_p R_q)$.

Graph models advantages: structural representation of functional blocks interactions and testability analysis execution possibility, because oriented graphs have expressed information threads, input and output vertices.

4.1. Testability values calculation for operational device (operational automata).

Testability analysis based on calculation of controllability, $C(R_q)$ and observability, $O(R_p)$.

Controllability calculation. Controllability of vertex $C(R_q)$ depends on controllability of previous vertex $C(R_p)$, and also on additional power of

instructions set $\frac{1}{k} \times \sum_{i=1}^k \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \right]$, of

activating arcs k , which enters in analyzed vertex $C(R_q)$. Here each of vertices contain of m operations, which initiate information transmission to $(R_p R_q)$.

Observability could be calculated similar to controllability calculation. Value of observability $O(R_p)$

oriented on analysis of arc-successors and outgoing arcs from vertex $O(R_p)$. Advantage of offered models is

universality of testability values oriented on direct and backward implication on graph, and their invariance to analysis and test synthesis of hardware and software components. Controllability $C(R_x) = 1$ of input nodes and observability $O(R_y) = 1$ of output nodes in graph are initiated by arithmetic “ones” values. Value $C(R_i) = 0$ has vertex, which could not be accessed by whatever path in graph. Practically, values of controllability lays in $[0;1]$ interval. For unconditional transfer arc weight is equal to one. In general case observability could be calculated by formula:

$$C(R_q) = \frac{1}{k} \times \sum_{i=1}^k \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \times C(R_p) \right]; \quad (4)$$

Observability calculation. Observability for graph vertices could be calculated as controllability values on direction from output vertices to input.

Observability estimation depends on observability value of preceding value, number of operations activated arc. Observability of end vertex $O(R_y) = 1$ or 100%. Observability could be equal to value from range $[0;1]$. $O(R_i) = 0$ for vertex, which is not observable on each of way in graph. Practically, all values are belongs to interval $[0;1]$. In general case, observability could be calculated using following formula:

$$O(R_p) = \frac{1}{k} \times \sum_{i=1}^k \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \times O(R_q) \right]; \quad (5)$$

Testability calculation.

$$T(R_i) = C(R_i) \times O(R_i) \quad (6)$$

Total testability could be calculated using following formula:

$$T_{\text{total}} = \frac{1}{n} \sum_{i=1}^n T(R_i) \quad (7)$$

where n – number of vertices.

Example 1. It is necessary to create software, which will allow solving simple arithmetic task $\varphi(x) + \omega(x)$, where

$$\varphi(x) = \begin{cases} x+3, & x > 2 \\ 2x-3, & 2 \leq x < 12 \\ -3x+7, & x \geq 12 \end{cases}, \quad \omega(x) = \begin{cases} \sin(x + \pi/3), & x < 2\pi\pi/ \\ \text{tg}(\pi g(+ 2, x \geq 2\pi\pi/ \end{cases}$$

Program 1.

```

1: #include <iostream>
2: #include <math.h>
3: using namespace std;
4: int main()
5: {
6:     const double Pi=3.14159;
7:     double F, w, f, x;
8:     cin>>x;
9:     if (x<2) f = x+3;
10:    else if ((x>=2) && (x<12)) f=2*x-3;
11:    else f=-3*x+7;
12:    if (x<2./3.*Pi)
13:    w=sin(x+Pi/3);
14:    else w=sin(Pi*x)+2;
15:    F=f+w;
16:    cout<<F<<endl;
17:    return 0;

```

Presented software include fault in one of arithmetical operators.

```

12: w=sin(x+Pi/3);
13: else w=sin(Pi*x) - 2;
14: F=f+w.

```

Task consists of detection of fault place in program code using testability analysis of graph model. Program code corresponds to circuit, presented on figure 4:

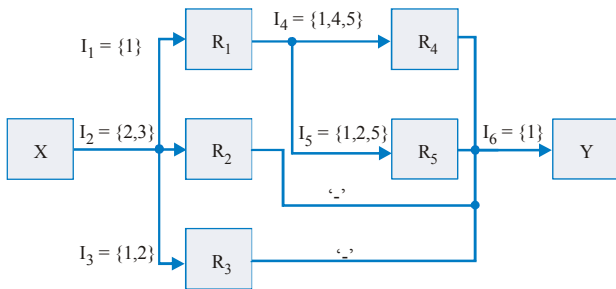


Figure 4. Register transfer graph

In braces on the figure pointed the arithmetical operations. So, there are:

- {1} – addition operation “+”;
- {2} – production operation “*”;
- {3} – subtraction operation “-”;
- {4} – dividing operation “/”;
- {5} – “sin” operation.

Most frequently used operations in code are summation (five times), production (four times), dividing (two times), “sin” operation (two times). Supposedly, fault could occur in code line, which contain summation {1} and production {2} operations.

For definition of additional code lines, which should be checked it is necessary to provide testability analysis.

Controllability values will be equal to:

$C(X)=1; C(R_1)=0,2; C(R_2)=0,4; C(R_3)=0,4;$
 $C(R_4)=0,12; C(R_5)=0,12; C(Y)=0,052.$

Observability values:

$O(R_1)=0,247; C(R_2)=0,2; C(R_3)=0,2; O(Y)=1;$
 $C(R_4)=0,2; C(R_5)=0,2; C(X)=0,069.$

Testability values:

$T(Y)=0,052; T(R_1)=0,024; T(R_2)=0,08; T(R_3)=0,08;$
 $T(R_4)=0,024; T(X)=0,069; T(R_5)=0,024.$

From TA is clear that vertex R₁, which activated by {1} operation (summation operation) has minimal (worst) value of controllability. Vertices R₂, R₃, R₄, R₅ with operations {2,3}, {1,2}, {1,4,5}, {1,2,5} have minimal (worst) values of observability. Vertices R₁, R₄, R₅ with operations {1}, {1,4,5}, {1,2,5} have minimal testability. Based on testability analysis it is necessary to check operations of summation, division, and “sin”.

In program code two lines with such operations could be selected, line 13 contains the desired fault:

```

12: w=sin(x+Pi/3);
13: else w=sin(Pi*x)+2.

```

5. Testability analysis for FSM

Here the main difficulty consist in necessary of checking all states in FSM, deadlocks and collisions, forks and feedbacks in code, (if, case, loop). Regard to control automata it is necessary to mark out following rule: each FSM should contain assertions, which allow to check states and transitions encoding [19].

Controllability, observability and testability values could be calculated similar to calculations for operational automata.

6. Strategy of control points selection and assertions placing in program code

General formula for control points selection procedure is following:

$$Z = \{Y_{TY}\} \cap \{A_{Rules}\} / \{Z_{TY}\},$$

where $\{Y_{TY}\}$ – set of points selected using TA method; $\{A_{Rules}\}$ – set of control points selected by rules of assertions usage; $\{Z_{TY}\}$ – set of points selected using TA method but could not be used for assertions usage.

Offered strategy of modification consist of separating test procedure on two modes – normal functioning and test circuit functioning. Additional conditional vertices could be added to analyzed graph to ensuring 100% observability or controllability. Expected, that such modification approach will significantly increase product testability with insufficiently hardware overheads.

7. Conclusions

Scientific novelty: new method of testability analysis for control and operational automata is offered. Methodology of test points selection and assertions placement is proposed. Such algorithms allow to increase fault coverage, total product testability and decrease test procedure time. Innovation technologies of testable design of software and hardware products, which oriented

on effective test cases development and verification, are considered.

Practical value of proposed methods and models: high interest of software companies in new test technologies for testing and verification of software products.

Advantages: 1. Assertions usage allows to decrease test time into 2-3 times; 2. Simple method of testability analysis [30-34].

Disadvantage: 1. Testability analysis on the gate level (for digital devices) could give more accurate analysis; 2. Additional test variables insert for software and hardware products testing.

8. References

1. *IEEE P1500/D11*. January 2005. Draft Standard Testability Method for Embedded Core-based Integrated Circuits. New York. 2005. 138 p.
2. *Abramovici M., Breuer M.A. and Friedman A.D.* Digital systems testing and testable design. Computer Science Press. 1998. 652 p.
3. *IEEE Std 1149.1-2001*. Standard Test Access Port and Boundary-Scan Architecture. New York. 2001. 208 p.
4. *IEEE Std 1149.4-1999*. IEEE Standard for a Mixed-Signal Test Bus. New York., 2000. 84 p.
5. *IEEE Std 1149.6-2003*. Standard for Boundary-Scan Testing of Advanced Digital Networks. New York. 2003. 139 p.
6. *Jutman A.* Shift Register Based TPG for At-Speed Interconnect BIST. Proc. of 24th International Conference on Microelectronics (MIEL'04) // Serbia and Montenegro. 2004. P. 751-754.
7. *C.Su, S.W.Jeng, Y.T.Chen* Boundary scan BIST methodology for reconfigurable systems // Proceedings of ITC'98. P. 774-783.
8. *W.Feng, F.J.Meyer, F.Lombardi.* Novel control pattern generators for interconnect testing. Proc. Int'l Symp. Defect and Fault Tolerance in VLSI Systems. 1999. P. 112-120.
9. *L. Jin.* The driver/receiver conflict problem in interconnect testing with boundary-scan. Proc of Asian Test Symposium (ATS'93). Beijing. 1993. p. 210-214.
10. *A. Hassan, V. Agarwal, B. Nadeau-Dostie, and J. Rajski.* BIST of PCB interconnects using boundaryscan architecture. IEEE Trans. Computer-Aided Design. Vol. 11. P. 1278-1287.
11. *Ulf Pillkahn.* Structural test in a board self test environment. Proc. IEEE Int. Test Conf. (ITC'2000). Atlantic City. NJ. USA. 2000. P. 1005-1012.
12. *W.H. Kautz.* Testing of Faults in Wiring Interconnects. IEEE Trans. Computers. Vol. 23. No. 4. 1974. P. 358-363.
13. *Stephen Pateras.* IP for Embedded Diagnosis. IEEE Design and Test of Computers. 2002. P. 46-56.
14. *S.G. Hemmady, T.L. Anderson, Y. Zorian.* Verification and Testing of Embedded Cores. Proc. Design SuperCon. On-Chip Design. 1997. P 119-122.
15. *Samvel Shoukourian.* A Unified methodology for Offline and Online Testing. IEEE Design & Test of Computers. 1998. P. 73-79.
16. *Sharshunov S.G.* Postroenie testov microptocessirov. 1. Obschaya model. Proverka obrabotki dannih //Avtomatika I telemehanika. 1985. №11. P.145-155. (on russian)
17. *Chipulis V.P., Sharshunov S.G.* Postroenie testov microptocessirov. 2. Proverka hraneniya i peredachi dannih //Avtomatika i telemehanika. 1986. №1. P.139-150. (on russian)
18. *Mark Glasser, Adam Rose, Tom Fitzpatrick, Dave Rich, Harry Foster* Advanced Verification Methodology Cookbook. Version 2.0. 2006.
19. *Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale* Verification Methodology Manual for System Verilog. 2006 Synopsys, Inc. and ARM Limited. 528 p.
20. *OVM Class Reference, Version 1.0.1*, February 2008, 286 p.
21. *Miro Samek, C/C++ Users Journals*, advanced Solutions for professional developers. 2003. 8p.
22. *Raimondas Lencevicius, Edu Metz* Performance Assertions for Mobile Devices // Proceedings of ISSTA'06. Portland, Maine, USA. 2006.
23. *Michael Pellauer, Mieszko Lis, Donald Baltus, Rishiyur Nikhil* Synthesis of Synchronous Assertions with Guarded Atomic Actions. 2005.
24. *Don Mills* Being Assertive With Your X (SystemVerilog Assertions for Dummies). SNUG San Jose. 2004.
25. *R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar.* The ForSpec Temporal Logic: A new temporal property-specification language. In Tools and Algorithms for Construction and Analysis of Systems // Lecture Notes in Computer Science. 2002. Vol. 2280. P. 211-296.
26. *C. Eisner, D. Fisman, J. Havlicek, McIsaac, and D. Van Campenhout.* The definition of a temporal clock operator // Proceedings of ICALP. Eindhoven: Springer-Verlag. 2003. P. 857-870.
27. *I. Beer, S. Ben-David, C. Eisner, and A. Landver.* RuleBase: An industry-oriented formal verification tool // Proc. Design Automation. New York: ACM Press. 1996. P. 655-660.
28. *T. Arons, E. Elster, L. Fix, S. Mador-Haim, M. Mishaeli, J. Shalev, E. Singerman, A. Tiemeyer, M. Y. Vardi, L. D. Zuck.* Formal Verification of Backward Compatibility of Microcode // Computer Aided Verification. 2005. Vol. 4. № 1. P. 185-198.
29. *Kulak E.N., Kaminska M.A., Wade Ghribi, Hassan Kteiman.* Modifikaciya wyfrovich shem s ispolzovaniem metoda analiza testopigodnosti TADATPG // Radioelectronica i informatika №4. 2005. P.60-68. (on russian)
30. *Bayraktaroglu Ismet, Orailoglu Alex* The Construction of Optimal Deterministic Partitionings in Scan-Based BIST Fault Diagnosis: Mathematical Foundations and Cost-Effective Implementations. IEEE Transactions on Computers. 2005. P. 61-75.
31. *R. A. Rutman,* Fault Detection Test Generation for Sequential Logic Heuristic Tree Search. IEEE Computer Repository Paper No. R-72-187, 1972.
32. *J. Grason* TMEAS - A Testability Measurement Program // Proceedings of 16th Design Automation Conf. 1979. P. 156-161.
33. *J. Grason and A. W. Nagel* Digital Test Generation and Design for Testability // Journal Digital Systems, Vol.5, No. 4. 1981. P. 319-359.
34. *K.P. Parker, E.J. McCluskey* Probabilistic Treatment of General Combinational Networks // IEEE Trans. on Computers. vol. C-24. No.6. 1975. P. 668-670.