Proc. Estonian Acad. Sci. Engin., 1995, 1, 1, 51–67
https://doi.org/10.3176/eng.1995.1.04

FAULT DIAGNOSIS IN VLSI DEVICES

Raimund UBAR

Tallinna Tehnikaülikooli Arvutitehnika Instituut (Department of Computer Engineering, Tallinn Technical University), Ehitajate tee 5, EE-0026 Tallinn, Eesti (Estonia)

Presented by L. Mõtus

Received 23 March 1995, accepted 1 June 1995

Abstract. This paper focuces on a new approach of hierarchical fault diagnosis for VLSI devices based on alternative graphs (AG), which allows a uniform description of VLSI designs at different representation levels and the use of the same general fault model for all levels. A uniform fault tracing technique based on AGs for all representation levels is developed. The technique is consistent with the multiple fault case, and the fault class considered is general rather than restricted to the traditional logical level stuck-at fault class. The main contribution of the paper is to minimize the number of signal observations needed for an exact localization of the fault site.

Key words: VLSI devices, hierarchical models, alternative graphs, stuck-at faults, fault diagnosis, fault simulation.

1. INTRODUCTION

The concept of fault diagnosis is a central issue both in prototype validation and in reliable computation. Once an error is observed, the faulty component responsible for that error must be located. As circuit density increases, the cost and time of fault diagnosis in VLSI devices will grow rapidly. Therefore, the development of efficient fault diagnosis methodology is of utmost importance.

Earlier work in the area of VLSI diagnosis was related to both artificial intelligence (AI) and non-AI–based approaches. The AI-based approaches are divided into two categories: the rule-based approach [1] and reasoning from the first principles [2]. Both methods are time-consuming. Some modifications to improve the efficiency of reasoning based fault diagnosis were suggested in [3]. In [4] very high speed integrated circuit hardware description language (VHDL) descriptions are used for fault diagnosis. The diagnosing procedure is basically similar to gate-level diagnosis, only the level of hardware description is higher and the fault model differs correspondingly. In [5] new functional fault models based on VHDL descriptions were introduced. The disadvantages of VHDL approaches are in the diversity of fault models and in the difficulty to use other fault

analysis methods than direct fault simulation. AI-based methods use traditionally the single-fault assumption, which is another disadvantage of the approach. An exception is the method presented in [6].

The most traditional non-AI–based diagnosis conception is using fault dictionaries [7]. Fault dictionary generation is the result of a fault simulation process. It is well-known that a complete fault simulation is practically impossible, hence the fault dictionary based fault diagnosis cannot be adequate. Some techniques and methods, generally used to avoid a complete fault simulation, are listed in [8]. In [9] a fault diagnosis system is described, based on the images of sets under the functions designed specifically to deduce logic values in circuits under multiple fault conditions. The approach used in this paper falls into the general category of effect–cause analysis [10]. In another approach, post-test fast foult simulation is used to diagnose faults in structured designs [11]. These approaches are implemented only for the class of combinational circuits.

This paper focuses on the diagnosis methodology which operates on the observed erroneous behaviour and on the structure of the system. The behaviour consists of the error(s) observed on the system output lines and specific values on the circuit input lines. By examining the error and the structure of the system, possible error sources can be determined. In a system, there is a definite flow of signals from the inputs to the outputs. These signals flow through the system components that modify the signal values according to a functional specification. When one of the components is faulty, the value that a particular signal takes on can become erroneous. The fault diagnosis is to determine the relationships between the components, and if faulty, the way they can affect the signal values. This effect–cause relationship will be created by backtracking faulty signals and represented by diagnostic trees where each node except leaves represents a signal error (an effect) and the successors of this node represent the possible causes of this error (Fig. 1). Based on this tree, the guided signal probing can be carried out. Alternatively, if internal probing is impossible, the fault diagnosis based on the intersection of different diagnostic trees (for the case of a single fault assumption) can be carried out. This paper discusses the problems of generation and minimization of diagnostic trees.

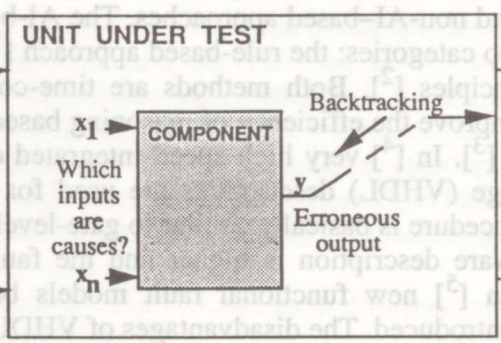


Fig. 1. Generation of the effect-cause relationship.

The paper contributes to the following areas. First, it introduces a new fault diagnosis approach based on alternative graphs (AG) [12], which allows a uniform description of VLSI designs at different representation levels and the use of the same general fault model for all levels. Using this approach, a three-stage hierarchical fault localization method was developed: 1) functional fault level diagnosis, 2) structural fault class level diagnosis, and 3) structural fault level diagnosis. This approach allows us to keep the complexity of candidate fault sets on each level as low as possible.

Another important contribution is the introduction of a uniform fault tracing technique based on AGs for all design representation levels. By this technique, a diagnostic tree (an effect–cause relationship between candidate faults [4]) is generated. The tree is equivalent to the traceback cone [11] where inconsistent faults are already eliminated. As different from [4], no fault simulation for extracting most of these faults is needed, and differently from [11], the traceback procedure will be carried out at a level higher than gate representation. The fault backtracking procedure is consistent with the multiple fault case. The fault class considered is general, not only restricted to the traditional stuck-at fault class.

The main contribution of the paper is to minimize the fault simulation amount compared to the known methods, and to minimize the number of signal observations (signal probing) needed for an exact localization of the fault site.

2. ALTERNATIVE GRAPHS AND FAULT PRESENTATION AT THE LOGICAL LEVEL

AG will be used here as a mathematical basis for fault backtracking in digital systems uniformly at different system representation levels. AGs were first proposed for test generation in digital circuits [13]. Unlike the analogical binary decision diagrams (BDD) described in [14], AGs support gate-level structural fault representation. Moreover, AGs are not restricted for use at the logical level only [15, 16].

AGs are a means for representing digital functions. First, consider a special case of the Boolean functions. An AG that represents a Boolean function is a directed noncyclic graph with a single root node, where all nonterminal nodes are labelled by (inverted or non-inverted) Boolean variables (arguments of the function) and have always exactly two successor-nodes whereas all terminal nodes are labelled by constants 0 or 1. For all nonterminal nodes, an one-to-one correspondence exists between the values of the label variable of the node and the successors of the node. This correspondence is determined by the Boolean function inherent to the graph.

Let us denote the variable which labels the node m by $x(m)$. We say that the value of the node variable activates the node output branch. According to the value of $x(m)$, one of two output branches of m will be activated. A path in an AG is called activated if all the branches that form this path are

activated. The AG is called activated to the value 0 (or 1) if there exists an activated path which includes both the root node and the terminal node labelled by the constant 0 (or 1). AG G_y with nodes labelled by variables $x_1, x_2, ..., x_n$, represents the Boolean function $y = f(X) = f(x_1, x_2, ..., x_n)$ if for each pattern of X, the AG will be activated to the value which is equal to y. We can consider a digital system as a network of components, each of which is described by one or more Boolean functions. Consequently, a digital system can be represented by a system of AGs. For the gate-level AG-description, the number of AGs is equal to the number of gates in the circuit. As an example, Fig. 2 shows a representation of a combinational circuit by AG. For simplicity, values of variables on branches are omitted (by convention, the right-hand branch corresponds to 1 and the lower-hand branch to 0). Also, terminal nodes with constants 0 and 1 are omitted (leaving the AG to the right corresponds to $y = 1$, and down – to $y = 0$).

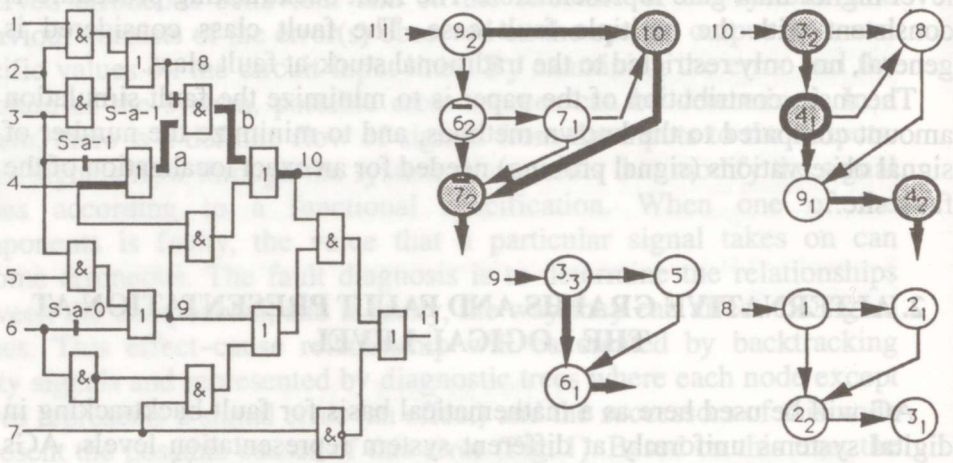


Fig. 2. Alternative graph representation of a combinational circuit.

Similar to the superposition of functions, the superposition of AGs can be defined [13]: if the label $x(m)$ of a node m in an AG G is a Boolean function which is represented by another AG $G_{x(m)}$, then the node m in G can be substituted by $G_{x(m)}$. Generation of an AG-model for a given gate-level digital circuit is based on the superposition of AGs. AGs for logical gates are assumed to be given as a source library. Starting from the gate-level AG-description and using iteratively the superposition procedure, we produce a more concise higher level representation of the circuit (by each substitution of a node with an AG, we reduce the model by one node and by one AG). As a result of the superposition procedure, we create structural AGs (SAG) which have the following property [13]: each node in a SAG represents a related signal path in the corresponding gate-level circuit. To avoid repetitive occurrence of the same subgraph in the AG-model, it is recommended to create separate AGs for tree-like subcircuits.

In this case, the number of all nodes in the set of SAGs will be equal to the number of paths in all tree-like sub-networks of the circuit. Hence, using the concept of SAGs, it is possible to ascend from the gate-level descriptions of digital devices to higher level structural descriptions without loosing the accuracy of representing gate-level stuck-at faults. The task of simulating structural stuck-at faults in a given path of a circuit can be substituted by the task of simulating faults at a node in the corresponding SAG. Figure 2 illustrates an example of a set of SAGs for a given circuit. The node 4_1 (bold circle) in the AG represents the upper (bold) path (4_1,*a*,*b*, 10) from the input branch 4_1 up to the node 10 in the circuit. The set of faults (4_1/1,*a*/1, *b*/1,10/1) related to the path is represented in the SAG by only one representative fault 4_1/1. One and only one node in the set of SAGs corresponds to each of the 17 signal paths in the tree-like subcircuits of the circuit (the upper fan-out branch is always denoted by the index 1 and the lower one by the index 2). The node variables are inverted if the number of invertors in the corresponding path of the circuit is odd.

Another way to generate logical level AGs is based on implementation-free descriptions of digital devices (Boolean expressions, truth tables, etc.). In this case, AGs do not differ from BDDs and we can use the methods developed for synthesis of BDDs [14]. Since AGs (or BDDs), obtained on the basis of functions only, do not represent the structure of the device, it is appropriate to name this class of AGs as functional AGs (FAG).

3. FAULT DIAGNOSIS AT THE LOGICAL LEVEL

3.1. Fault backtracking on alternative graphs

Each path in an AG describes the behaviour of the circuit in a specific mode of operation. The faults having effect on the behaviour are related to the nodes along a given path. A fault causes an incorrect leaving the path activated by a test. Hence, if we activate a path in an AG by a test pattern which fails, then all faults related to the nodes of the path can be regarded as fault candidates for the diagnosis procedure.

If an erroneous signal is detected in an output *y* of the circuit, then by fault backtracking procedure, the set of candidate faults which can explain the misbehaviour of *y*, will be created. This set will be represented in the form of a diagnostic tree (DT) with the root labelled by the failed output *y*. In this procedure, first, the path activated by a failed test pattern will be determined in the graph G_y. All the nodes of this path will be put into DT as successors of the root node. For each successor with a label x that corresponds to an internal node of the circuit and is represented by graph G_x, again an activated path will be determined whose nodes will form the set of successors of x in the DT. This procedure will be repeated recursively until all the leaves in the DT are labelled by input variables. The number of nodes in the DT found in this way is generally less than the

number of nodes contained in the traceback cone [11] (or transitive fan-in [4]) in the corresponding circuit.

With each node in the DT labelled by a variable x, we associate a representative fault $x/\neg D$ (x stuck-at $\neg D$) where $\neg D \in \{0,1\}$ is the opposite value to what x has in the test pattern. This fault represents all faults that can explain the signal failure along the path in the circuit represented by the node x in the AG.

As an example, Fig. 3 shows the diagnostic tree created by fault backtracking for the given test pattern and for the failure at the output 11 of the circuit given in Fig. 2. Each fault in the DT represents a whole set of faults in the circuit. For example, for the stuck-at 1 fault case, the fault 4_1/1 defined in the AG, represents faults 4_1/1, *a*/1, *b*/1, 10/1 in the circuit.

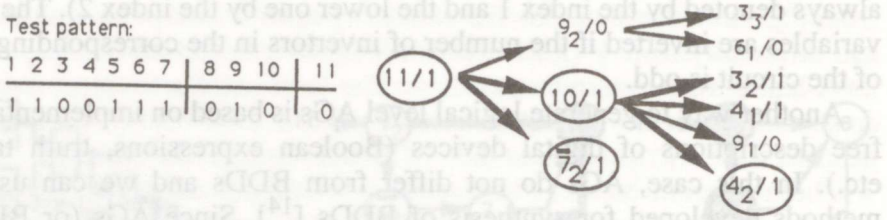


Fig. 3. Fault backtracking in alternative graphs.

3.2. Fault candidate tree reduction

In a general case, a DT is redundant in a sense that not all candidates in the DT built by fault backtracking have to be consistent to explain the misbehaviour of the circuit. There are three possibilities to exclude inconsistent candidates from a DT: using a special "*direction rule*" [17] defined for SAGs, using local fault simulation and using global fault simulation. But only one of them – the use of "direction rule" does not contradict the multiple fault case. Fault simulation can be used only for a simplified case of single fault assumption.

The "direction rule" was introduced on the basis of special properties of SAGs [17] and can be formulated by the following theorem.

T h e o r e m 1. *In an activated path of an AG G_y, which terminates in a node m^T, only these nodes m are consistent to explain the faulty value of y for which $z(m) = z(m^T)$ holds.*

The proof of the theorem is given in [17].

In the graphical interpretation of the theorem, only these nodes for which the *direction of leaving the node* along the activated path coincides with the *direction of leaving the graph* (terminal node) are consistent. As an example, let us look at the paths activated in AGs in Fig. 2 by the test pattern depicted in Fig. 3. From the "direction rule", it follows that the node 9_2 in G_{11} (consequently, with all its successors) and the node 9_1 in

G_{10} are to be excluded from the DT. The resultant DT is stressed by bold print.

Local fault simulation, acting only in the area of a given graph G_y, determines if the faults of nodes in the graph are able to change the value of y or not. It is equivalent to restricted fault simulation only in the scope of tree-like subcircuits. Local simulation can be used for nodes which remain in the DT after the action of the "direction rule". As an example, by local fault simulating it is possible to exclude additionally nodes 3_2 and 4_1 from the DT in Fig. 3, because the faults at these nodes cannot effect the value of 10. The resultant DT is shown by circles. It should be noticed that for the multiple case, this DT will not be valid any more. For example, this test pattern is able to detect the multiple fault (3_2/1, 4_1/1), however, after the fault simulation both of these faults are missing in the DT.

Global fault simulation acts in the scope of the whole circuit and has the goal to analyse the fault propagation through reconvergent fan-out regions. For the DT, it means that only these variables contained in the DT which represent stems of reconvergent fan-out regions are to be fault simulated globally.

3.3. Complexity considerations

Consider the DT generation separately for the multiple fault and single fault assumption cases. For the multiple fault case, the method of fault backtracking combined with the use of "direction rule" without fault simulation is valid. The idea of the method is closely related to the fault list generation procedure described in [11], except that in [11] gate level faults and the gate-level structure are used whereas in this paper, only fault class representatives and a higher macro-level are considered. The parity and signal value consistency calculations along each gate-level path are substituted in the present paper by the consistency proof only for the representative fault of the path. Hence, the complexity of the present method is lower than that of [11]. For example, the 66 stuck-at faults of the circuit in Fig. 2 are represented in the AG-model by 34 representatives only. Comparison with [4] is not possible because the fault candidate generation from the transitive fan-in occurs in [4] only by simulation.

For a single fault assumption, the fault simulation for a reduced candidates set is accepted. In [4] all faults of the transitive fan-in area are to be fault simulated whereas in the present paper, only faults which have remained in the DT after using the "direction rule" are to be simulated. This explains the complexity reduction achieved by the present method compared to [4]. For example, for the circuit in Fig. 3, the transitive fan-in consists of 33 stuck-at faults which all need simulation according to [4]. Using the presented AG-approach, for the test pattern in Fig. 3, only five representative faults are to be simulated.

3.4. Fault localization at the logical level

The diagnostic tree generated by fault backtracking is the guide for fault diagnosis. Since the tree consists of fault class representatives, the diagnosis will be processed in two steps: 1) fault class localization and 2) fault site localization.

First, let us consider the multiple fault assumption. For multiple fault diagnosis, only guided probing technique is discussed here. If a multiple fault will cause the misbehaviour of the circuit, then at least one of the component faults has to belong to the DT created by the procedure described in section 3. Consequently, this fault has to be detected since the probing procedure follows the DT. If a fault is detected, the multiple fault analysis has to be carried out. First, the simulation of the detected fault will be produced. If the fault is classified as detectable, then the fault diagnosis for the given test pattern will terminate. Otherwise, if the fault is classified as not detectable, then the multiple fault analysis will continue. Next, the detected fault will be inserted into the model and a new fault conditioned DT for the given test pattern will be created. This DT will guide the fault diagnosis for the next component of the multiple fault. If the representative fault(s) is (are) detected, the fault site localization among the related fault class(es) will begin.

As an example, suppose that the fault $3_2/1$ will be detected guided by the DT in Fig. 3. A new activated path in G_{10} conditioned by this fault will traverse only one additional node 8. As there are no other alternatives, no probing is needed any more and the fault 8/1 will be diagnosed as the second component of the multiple fault. For the second diagnosis step, these two faults create the following fault candidates: $3_2/1 \Rightarrow \{3_2/1, a/1, b/1, 10/1\}$ and $8/1 \Rightarrow \{8/1, b/1, 10/1\}$. The information about the common parts of these fault sets can be used in diagnosing the fault site.

In the single fault assumption, the fault diagnosing procedure analogous to [11] can be used only if all representative faults in DTs are decoded into plain fault lists. As a result, diagnosis at the fault class level may give a multiple fault case. After intersection of the corresponding fault sets, a single fault site diagnosis can be made.

4. ALTERNATIVE GRAPHS AND FAULT REPRESENTATION AT THE FUNCTIONAL LEVEL

Consider a complex digital device in a general case as a dynamic system $S = (Z,F)$, where Z is a set of digital variables z (the number of values of z is arbitrary but finite) and F is a set of arbitrary digital functions on Z. The system S can be represented by an AG-model where nodes are labelled by functions $f \in F$, variables $z \in Z$ or digital constants and an one-to-one correspondence exists between the values of the node label and the successors of the node.

Depending on the class of digital systems (or level of their representation), we can classify different classes of AGs. For example, for register transfer level (RTL), representations of digital systems consisting of control and data paths, internal nodes in AGs are labelled by control variables of Z (or control functions of F) whereas terminal nodes are labelled by data functions of F (or constants). Here, the following interpretation can be established: terminal nodes represent the data part and internal nodes describe the control part of the system. For logical level representations, all nodes are labelled by logical functions (variables or constants). To create concise descriptions for structured random logic, mixed-level representations are possible. For example, in finite state machines, alongside with Boolean variables, integer state-variables and integer constants for representing states can be introduced. In such a way, AGs serve as a universal means to describe digital systems at different representation levels. Hierarchical multilevel descriptions are easy to produce since implementation details of functions given as labels at the nodes of higher-level AGs can be represented by additional lower-level AGs. Methods of generating such AGs are described in [12].

An example of a digital system consisting of a data part and a control part is depicted in Fig. 4. Figure 5 shows a set of AGs representing the system. The data part is represented by AGs for the output data variable *OUT*, bus variables M_1 and M_2, register variables $R_0, ..., R_n$, functional blocks F_1, F_2, F_3, and flag variables x_1, x_2. Only the graphs for flag variables are binary, other graph variables are integers. Their values are determined by expressions in the terminal nodes of the corresponding AGs. Nonterminal nodes of the AGs of the data part are labelled by clock variables C_1, C_2 and control variables $y_1, ..., y_7$ which represent different subfields of the control field Y in the microinstruction word. The control part is represented by AGs for the microinstruction variable *MIR*, *ROM* address variable A, and the control variable Z for multiplexing address sources. The system is synchronized by the two clocks C_1 and C_2. The graphs with clock variables represent the behaviour of the corresponding sequential circuit in the recursive form where the apostrophe denotes a delay (the value of the variable with apostrophe is related to the time frame before clocking). A hierarchical representation is easy to organize. The following possibilities for descending from higher levels to lower ones can be highlighted: both the integer graph-variables and the integer-variables in nonterminal nodes can be vectorized and substituted by concatenation of field-variables (for example, bit-variables), and the terminal nodes can be substituted by lower level graphs which represent implementation details of the corresponding functions more precisely [12].

Different fault models defined at different representation levels of digital systems are replaced on AGs by a uniform node fault model: 1) the output branch of a node is always activated (stuck-at value), 2) the branch is broken (stuck-open), and 3) instead of the given branch another branch or a set of branches is activated. The stuck-at value fault model used for gate-level circuits corresponds to faults of nodes labelled by the Boolean variables. The functional fault model introduced in [18] for the control part

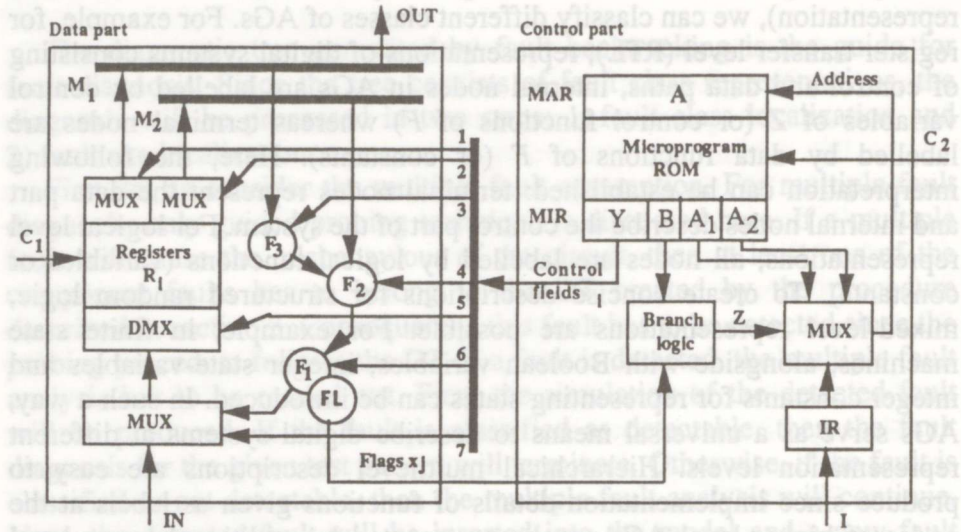


Fig. 4. A digital system represented at the register transfer level.

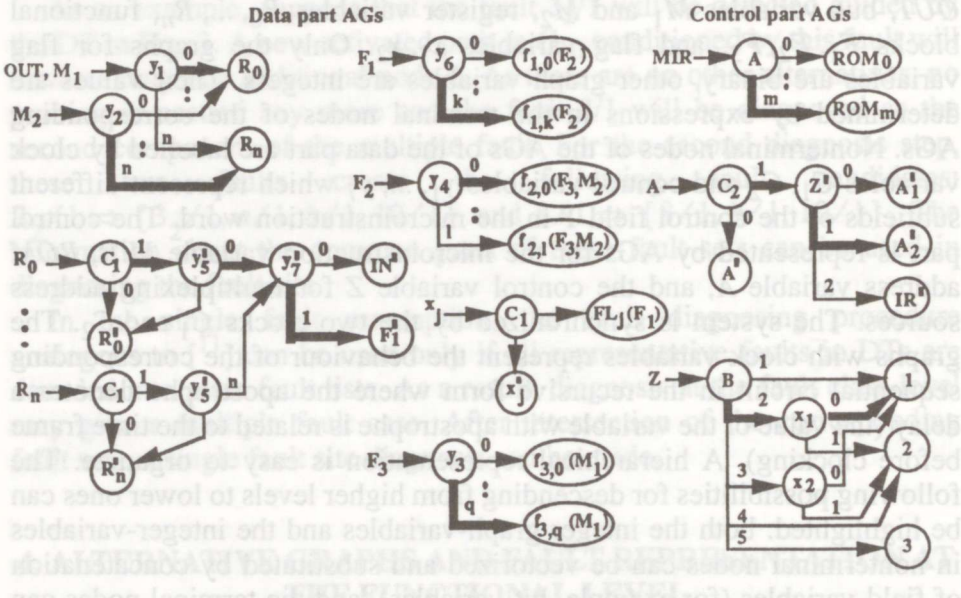


Fig. 5. Alternative graph representation of the digital system.

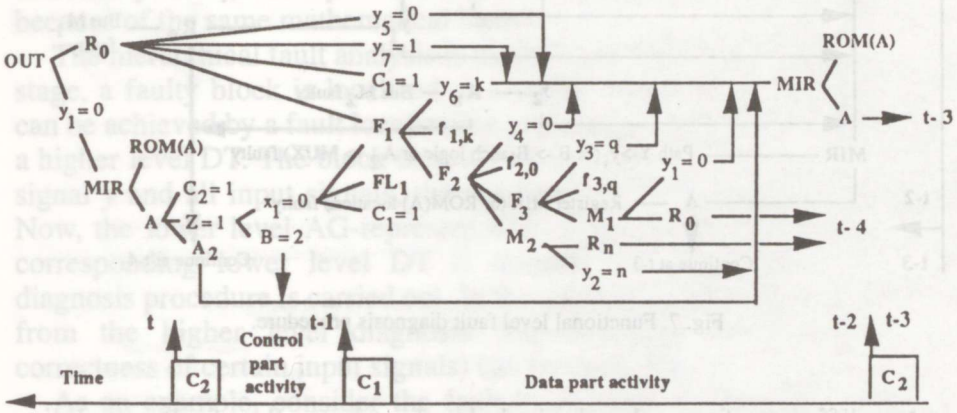
of microprocessors is covered by faults of nodes labelled by instruction word variables. The fault model defined for AGs covers also the fault classes introduced in [5] for VHDL descriptions.

Each path in an AG describes the behaviour of the system in a specific mode of operation. The faults having effect on the behaviour are related to the nodes along the given path. A fault causes an incorrect leaving the path

activated by a test. The physical meaning of the faults associated with node outputs depends on the physical meaning of the node. Depending on the adequacy of representing the structure of the system, the fault model proposed can cover a wide class of structural and functional faults introduced for digital circuits and systems. The fault model defined on AGs can be regarded as a generalization of the classical gate-level stuck-at fault model - the latter is defined for Boolean variables, the former for the nodes of AGs.

5. FAULT DIAGNOSIS AT THE FUNCTIONAL LEVEL

The fault backtracking procedure and the creation of the DT will be carried out at the higher level AGs as described in section 3 for the logical level AGs. However, in the DT nodes, instead of representative faults, simply variables with their expected (simulated) values will be marked. As an example, Fig. 6 demonstrates the DT for a given test sequence and the failure at the data output at the time moment *t*. From this tree, the fault localization procedure in Fig. 7 results.



Test sequence

Time	C1	C2	A	y1	y2	y3	y4	y5	y6	y7	B	A1	A2	R0	Rn	x1	x2
t - 2		1	a	0	n	q	0	0	k	1	2		m	r01	rn		
t - 1	1			0	n	q	0	0	k	1	2		m	r02		0	0
t		1	m	0										r02		0	0

Fig. 6. Diagnostic tree for the given test sequence.

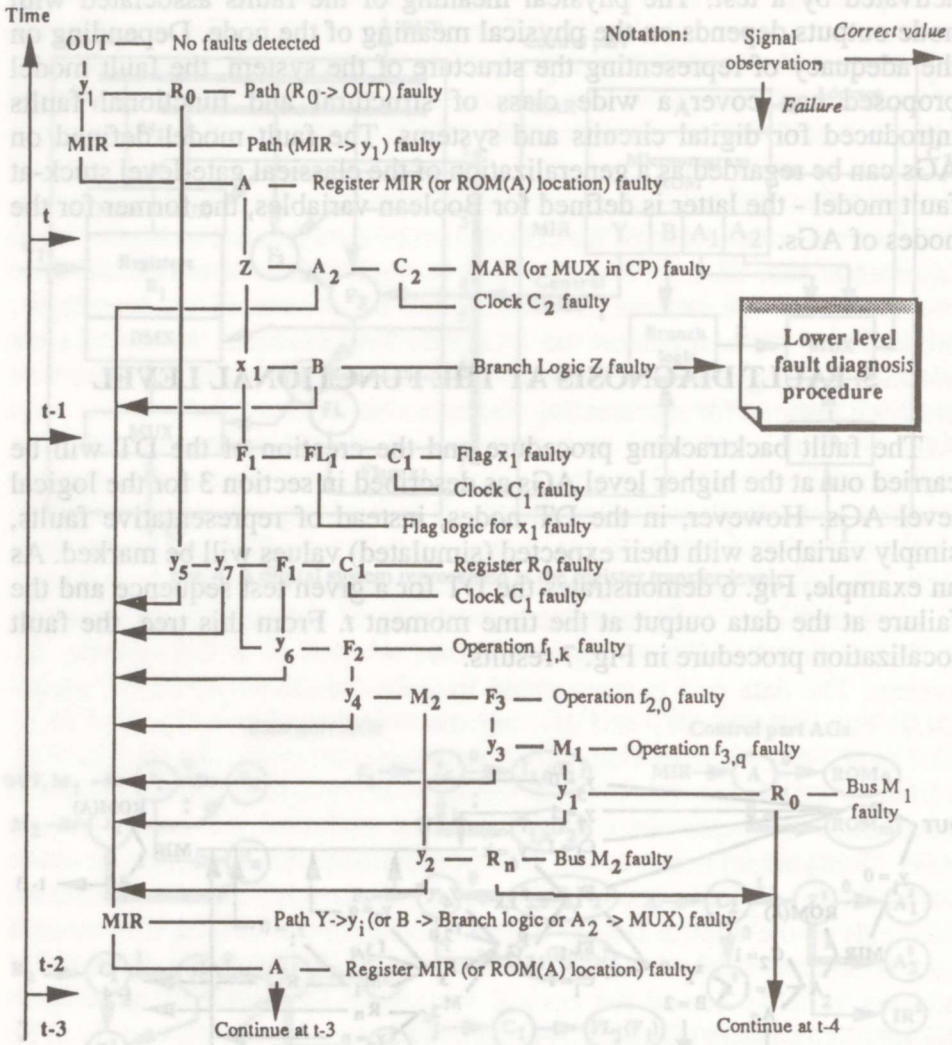


Fig. 7. Functional level fault diagnosis procedure.

As different from the logical level, where the fault resolution possibilities based on differentiating only two signal values are very restricted, at the functional level, more powerful fault diagnosis analysis based only on simulation (and not on signal probing) is possible. Compared to the fault search at the logical level where, in a general case, all the nodes in the DT are pretenders for signal probing, at the functional level, many nodes in the DT can be excluded, based on the current signal observation results. The following theorem can be formulated.

T h e o r e m 2. *Consider an AG G_y where y has an erroneous value y*, the path L in G_y is activated by the failed test pattern and the variables z(m), where $m \in L$, are the successors of y in the DT. The variable z(m), $m \in L$, cannot cause an error in y if there is no path in G_y from m up to a terminal node m' where $z(m') = y^*$ holds.*

Consider an AG G_y where a path L is activated. Using Theorem 2 it is easy now to determine the possible causes of the erroneous value of y. Only these successors $z(m)$ of y in DT can cause an error (and be a pretender for further signal probing), for which a path in G_y from m up to a terminal node m' with $z(m') = y^*$ exists. For example, let us look at the DT in Fig. 6. Suppose the variable OUT was observed and an erroneos value OUT^* was detected. If in G_{OUT} a terminal node with $R_i = OUT^*$, $i \neq 0$ exists, then it results that y_1 has a faulty signal $y_1 = i$ (instead of $y_1 = 0$) and no additional observation of y_1 by probing is not necessary. Otherwise, if no such a terminal node with $R_i = OUT^*$, $i \neq 0$ exists, the variable y_1 should be correct and the register R_0 has to be the cause of the error.

In such a way, with a functional level fault diagnosis procedure organized on the basis of DT, many decisions about the state of signals (faulty or not faulty) can be made not by probing signals but only through the corresponding AG analysis.

6. HIERARCHICAL FAULT DIAGNOSIS

Based on the AG representation of a digital system, a hierarchical fault analysis can be easily carried out. The main feature of the approach is the uniformity of the analysis methods at different representation levels because of the same mathematical basis.

The hierarchical fault analysis is made in the following way. At the first stage, a faulty block is localized at a higher functional level. This result can be achieved by a fault localization procedure organized on the basis of a higher level DT. The block is qualified as faulty if it has a faulty output signal y and all input signals, the successors of y in the DT, are correct. Now, the lower level AG-representation of the block is made taken, the corresponding lower level DT is created, and the lower level fault diagnosis procedure is carried out. In this procedure, if possible, the results from the higher level diagnosis (for example, the information of correctness of certain input signals) can be exploited.

As an example, consider the fault localization procedure, carried out along the tree in Fig. 7. Suppose that the Branch logic Z is qualified as faulty. In the corresponding higher level AG G_Z for the faulty block, a path ($B = 2, x_1 = 0, 1$) is activated. But no errors in B and x_1 (the successors of Z in the DT) are detected. The gate-level circuit of the Branch logic with the corresponding higher and lower level AGs and the failed test pattern related to this block are depicted in Fig. 8. From the translation of the higher level failure $Z = 1 \Rightarrow Z = 3$ to the lower level $(Z_1.Z_2) = (0.1) \Rightarrow (Z_1.Z_2) = (1.1)$, it follows that the Branch logic for the first bit of the word Z is faulty. The path activated by the failed test pattern at the lower level AG $G_{Z,1}$ is illustrated in Fig. 7 by bold print. It consists of nodes B_{11}, $\neg B_{12}$, B_{22}, $\neg B_{31}$, x_1, x_2 which all will be put in the DT as successors of the faulty variable Z_1 (Fig. 9). From these nodes, x_1 and x_2 can be excluded as a result of the higher level fault analysis

(Theorem 2): from the node x_1 in G_Z no path exists to the terminal node 3 (3 is the faulty value of Z), but x_2 is missing at the higher level activated path and, therefore, can not cause the erroneous signal Z. Further, using the "direction rule" (Theorem 1), the nodes $\neg B_{12}$, B_{22}, $\neg B_{31}$ can also be excluded from the DT. As a result, the DT will contain only node B_{11} which can now be automatically (without probing) qualified as faulty. Finally, the multiple fault analysis follows, as explained in section 3.4. From this analysis, the fault of the node $\neg B_{21}$ results. Both faults $B_{11}/1$ and $\neg B_{21}/1$ are shown in the corresponding Branch logic circuit in Fig. 9.

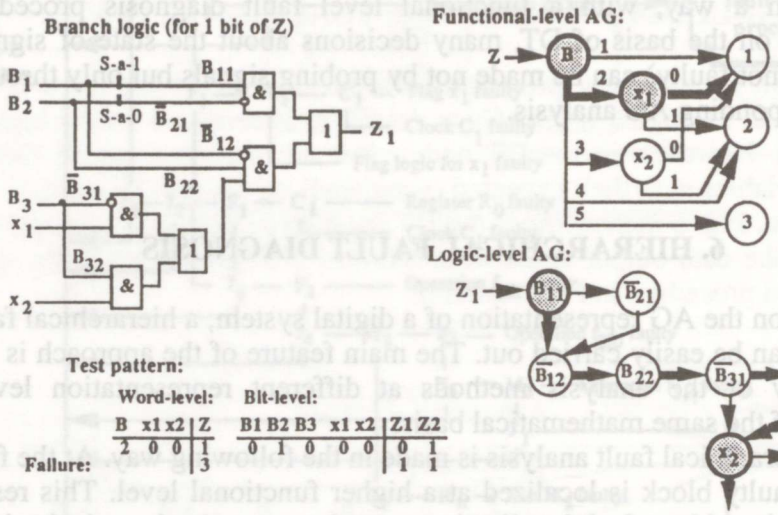


Fig. 8. Hierarchical design and test representation.

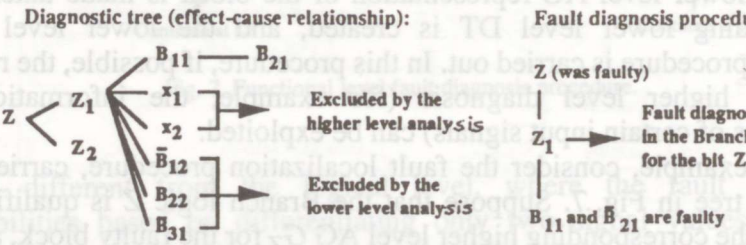


Fig. 9. Lower level diagnostic tree and fault diagnosis procedure.

It is worth mentioning that by fault simulation (for the case of single fault assumption), the fault $B_{11}/1$ should be excluded from the DT as well, thus leaving the DT empty. Hence, using the simulation-based approaches for determining fault candidates, like [4] and [11], a more precise fault diagnosis in the Branch logic would have been impossible.

7. CONCLUSIONS

The paper presents a new conceptual approach based on alternative graphs for fault diagnosis in digital systems. For the first time, a hierarchical approach is developed based on the uniform mathematical apparatus for each level of hierarchy. This approach facilitates a uniform fault analysis at all levels and easy transportation of diagnosis results obtained at a higher design level to a lower one to continue the candidate fault set reduction. Also, it allows us to keep the complexity of candidate fault set at each level as low as possible. The fault model defined in the alternative graphs is the same for each design representation level, except the interpretation the model has in relation to different levels. It was also shown that in some cases the multi-level diagnosis can offer more adequate results than the diagnosis carried out only at the logic level.

Compared to the known methods, the approach presented allows us to reduce the amount of fault simulation for the candidate fault set reduction, and to reduce the number of signal observations for fault site exact localization.

ACKNOWLEDGEMENTS

The support by the Estonian Science Foundation grant No 574 and by the fellowship from C.I.E.S., France, is greatly appreciated.

REFERENCES

1. Grillmeyer, O. and Wilkinson, A. J. The design and construction of a rule base and an inference engine for test system diagnosis. – In: IEEE Int. Test Conf., 1985, 857–867.
2. Davis, R. Diagnostic reasoning based on structure and behaviour. – Artif. Intell., 1984, **24**, 347–410.
3. Thearling, K. H. and Iyer, R. K. Diagnostic reasoning in digital systems. – In: 18th Int. Symp. Fault Tolerant Computing, Tokyo, June 1988, 286–291.
4. Pitchumani, V., Mayor, P., and Radia, N. Fault diagnosis using functional fault model for VHDL descriptions. – In: IEEE Int. Test Conf., Nashville, Oct. 1991, 327–337.
5. Ward, P. C. and Armstrong, J. R. Behavioral fault simulation in VHDL. – In: 27th ACM/IEEE Design Automation Conf., 1990, 587–593.
6. Kleer, J. and Williams, B. C. Reasoning about multiple faults. – In: Nat. Conf. Artif. Intell., Philadelphia, 1986.
7. Ramamoorthy, C. V. A structural theory of machine diagnosis. – In: Proc. Spring Joint Comp. Conf., 1967, 743–756.
8. Marzouki, M., Laurent, J., and Courtois, B. Coupling electron-beam probing with knowledge-based fault localization. – In: IEEE Int. Test Conf., Nashville, Oct. 1991, 238–246.
9. Rajski, J. GEMINI – a logic system for fault diagnosis based on set functions. – In: 18th Int. Symp. Fault Tolerant Computing, Tokyo, June 1988, 292–297.
10. Abramovici, M. and Breuer, M. A. Multiple fault diagnosis in combinational circuits based on an effect-cause analysis. – IEEE Trans. Comput., 1980, **C-29**, 451–460.
11. Waicukauski, J. A., Gupta, V. P., and Patel, S. T. Diagnosis of BIST failures by PPSFP simulation. – In: 18th IEEE Int. Test Conf., Washington, Sept. 1987, 480–484.

12. Убар Р. Р. Альтернативные графы и техническое диагностирование дискретных объектов. – Электронная техника, 1988, **8**, 5 (132), 33–57.
13. Убар Р. Р. Генерирование тестов для цифровых схем при помощи модели альтернативных графов. – Тр. Таллин. политехн. ин-та, 1976, 409, 75–81.
14. Akers, S. B. Binary decision diagrams. – IEEE Trans. Comp., 1978, **27**, 509–516.
15. Ubar, R. Vektorielle alternative Graphen und Fehlerdiagnose für digitale Systeme. – Nachrichtentechnik/Elektronik, 1981, **31**, 1, 25–28.
16. Ubar, R. Test pattern generation for digital systems on the vector AG-model. – In: 13th Int. Symp. Fault Tolerant Computing, Milano, Italy, 1983, 347–351.
17. Убар Р. Р., Эвартсон Т. А. Оптимизация процессов поиска неисправностей в типовых элементах замены цифровых систем. – In: Автоматизация конструкторского проектирования в радиоэлектронике и вычислительной технике, **1**. Вильнюс, 1981, 175–184.
18. Thatte, S. M. and Abraham, I. A. Test generation for microprocessors. – IEEE Trans. Comput., 1980, **29**, 429–441.

SUURTE INTEGRAALSKEEMIDE RIKETE DIAGNOSTIKA

Raimund UBAR

On tutvustatud uut, alternatiivsetel graafidel põhinevat, hierarhilist suurte integraalskeemide rikete diagnoosimeetodit. Alternatiivsed graafid võimaldavad integraalskeemi eri esitustasanditel ühtselt kirjeldada, kasutada sama matemaatilist aparaati, samu analüüsivahendeid ja üldist rikete mudelit. Seesugune lähenemine lubab rikete analüüsi teha skeemi eri abstraktsiooniastmetel üht moodi. Ühtlasi saab kõrgemal tasandil leitud diagnositulemusi kergesti üle kanda madalamale tasandile edasiseks rikete "kandidaatide" hulga minimeerimiseks. See võimaldab hoida rikete "kandidaatide" keerukuse igal tasandil nii väikese kui võimalik. Rikete mudel, mis defineeritakse alternatiivsetel graafidel, jääb samaks skeemi iga kirjeldustasandi puhul, erinev on ainult mudeli interpretatsioon. Uus meetod võimaldab minimeerida rikete simuleerimise mahtu võrreldes traditsiooniliste meetoditega ja minimeerida rikke täpse asukoha määramiseks vajalike signaalide mõõtmiste arvu.

ДИАГНОСТИКА НЕИСПРАВНОСТЕЙ В БОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМАХ

Раймунд УБАР

Представлен новый, основанный на альтернативных графах иерархический метод диагностики неисправностей в больших интегральных схемах. Альтернативные графы позволяют описывать интегральные схемы единообразно, используя один и тот же математический аппарат, одни и те же средства анализа и универсальную модель неисправностей на всех уровнях представления схемы. Более того, выявленные на высшем уровне результаты диагноза легко переносимы на следующий нижний

уровень, что позволяет свести число "кандидатов" неисправностей на каждом последующем уровне до минимума. Основанная на альтернативных графах модель неисправностей остается неизменной на каждом уровне описания схемы, меняется лишь ее интерпретация.

Преимущество предложенного метода перед традиционными заключается в том, что удается минимизировать не только объем моделирования неисправностей, но и число измеряемых сигналов, а следовательно, быстрее определить точное место неисправности.