Decision Diagrams and Digital Test

R. Ubar
Computer Engineering Department, Tallinn Technical University
Raja 15, Tallinn, Estonia, raiub@pld.ttu.ee

Abstract: Short overview about the problems and tasks of digital test is given. The most important question in testing today's complex digital systems is: how to improve the testing quality at continuously increasing complexities of systems? Two main trends can be observed: defect-orientation to increase the quality of testing, and high-level modelling to reduce the complexity problems of diagnostic analysis. Both trends can be joined in the hierarchical approach. Decision Diagrams (DD) serve as a good tool for hierarchical modelling and diagnostic analysis of digital systems. Traditional Binary Decision Diagrams are well known for working with logic level. New generalizations of BDDs in a form of High-Level DDs and Vector DDs as efficient tools for test generation and fault simulation of complex digital systems are discussed in the paper.

I. INTRODUCTION

Testing and diagnosis of digital electronics systems have faced a lot of problems produced mainly by the complexity of systems. Test generation encompasses three main activities: selecting a model of the system, developing a fault model, and generating tests to detect all the faults covered by the accepted fault model. The efficiency of test generation (quality of tests, speed of test generation) is highly depending on the methods used for modelling the system behaviour and its faults.

The most important question in testing today's complex digital systems is: how to improve the testing quality at continuously increasing complexities of systems? Two main trends can be observed when searching solutions for the formulated problem: defect-orientation, and high-level modelling.

Traditional low-level test generation methods and tools for complex digital systems have lost their importance, other approaches based mainly on higher level functional and behavioral methods are gaining more and more popularity [1-3]. However, the trend towards higher level modelling moves us even more away from the real life of defects and, hence, the accuracy of testing decreases. To handle adequately defects in deep-submicron technologies, new fault models and defect-oriented test methods should be used. But, the defect-orientation is increasing even more the complexity of test problems. To get out from the deadlock, these two opposite trends – high-level modelling and defect-orientation – should be combined into hierarchical approach. The advantage of the hierarchical approach compared to the high-level functional modelling lies in the possibility of working with coarse test plans on higher functional levels to achieve high speed in simulation, and modelling faults on more detailed lower levels to achieve high accuracy.

The drawback of traditional multi-level and hierarchical approaches to digital test lies in using different languages and models for different levels. Most frequent examples are logic expressions for combinational circuits, state transition tables and diagrams, or microprogram languages for finite state machines (FSM), abstract execution graphs, system graphs, instruction set architecture descriptions, flow-charts, hardware description languages (HDL, VHDL, Verilog), Petri nets for system level descriptions etc. All these models need different manipulation algorithms and fault models which are difficult to couple together for implementing hierarchical test generation or fault simulation algorithms.

Better uniform opportunities for hierarchical diagnostic modelling of digital systems provide Decision Diagrams (DD) [4-10]. Binary Decision Diagrams (BDD) have found already very broad applications in logic design as well as in logic test [4-6]. A special class of BDDs called Structurally Synthesized BDDs (SSBDD) allow to represent directly in the graph gate-level structural faults [7]. Recent research has shown that generalization of BDDs in the form of high level DDs provide a uniform model for both gate and register transfer level or even behavioral level test generation [8-10].

The disadvantage of the traditional approaches to test is also the use of gate-level SAF model. The SAF model which has been rather popular in test quality estimating has not withstood the test of time. It has been shown that high SAF coverage cannot quarantee high quality of testing [11-12]. The reason is that the SAF model ignores the actual behavior of deep-submicron circuits, and does not adequately

represent the majority of real IC defects and failure mechanisms which often do not manifest themselves as stuck-at faults. To improve the test quality, there is a need to replace abstract fault models like SAF with realistic defect models.

The types of faults that can be observed in a real gate depend not only on the logic function of the gate, but also on its physical design. These facts are well known [13-15] but usually, they have been ignored in engineering practice. In earlier works on layout-based test techniques [14,15] a whole circuit having hundreds of gates was analysed as a single block. Such an approach is computationally expensive and highly impractical as a method of generating tests for real VLSI designs.

To handle physical defects in fault simulation, we still need logic fault models for the following reasons: to reduce the complexity of simulation (many physical defects may be modelled by the same logic fault), a single logic fault model may be applicable to many technologies, logic fault tests may be used for physical defects whose effect is not well understood. But the most important reason for logical modelling of physical defects is to get a possibility for moving from the lower physical level to the higher logic level which has less complexity.

In this paper, we present, first, in Section 2 a method for modelling physical defects by generic Boolean differential equations which gives a possibility to map the defects from the physical level to the logic level. A generalization of the SAF model called Functional Fault Model (FFM) is presented in Section 3. It will be shown also how the FFM can be retreated as a uniform interface for mapping faults from a given arbitrary level of abstraction to the next higher level in test generation or fault simulation processes. For hierarchical diagnostic modeling DDs are used. In Section 4 structurally synthesized BDDs are discussed for logic level test generation and fault simulation. Section 5 explains how the hierarchical approach can be implemented by using higher level and vector DDs. Section 8 with general remarks concludes the paper.

2. MODELLING DEFECTS AND FAULTS

Consider a Boolean function $y = f(x_1, x_2, ..., x_n)$ implemented by an embedded component C in a circuit. Introduce a Boolean variable d for representing a given physical defect in the component, which may affect the value y by converting the Boolean function f into another function $y = f^d(x_1, x_2, ..., x_n)$ where in fact, some of the arguments x_i can fall out, simplifying in that way the function because of the fault. Introduce for the block C a generic parametric function

$$y^* = f^*(x_1, x_2, ..., x_n, d) = \bar{d} f \vee d f^d \qquad (1)$$

as a function of a defect variable d, which describes the behavior of the component simultaneously for both, fault-free and faulty cases. For the faulty case, the value of the defect variable d as a parameter is equal to 1, and for the fault-free case $d = 0$. In other words, $y^* = f^d$ if $d = 1$, and $y^* = f$ if $d = 0$. The solutions of the Boolean differential equation

$$W^d = \frac{\partial y^*}{\partial d} = 1 \qquad (2)$$

describe the conditions which activate the defect d on a line y. The parametric modeling of a given defect d by equations (1) and (2) allow us to use the constraints $W^d = 1$, either in defect-oriented fault simulation, for checking if the condition (2) is fulfilled, or in defect-oriented test generation, to solve the equation (2) when the defect d should be activated and tested.

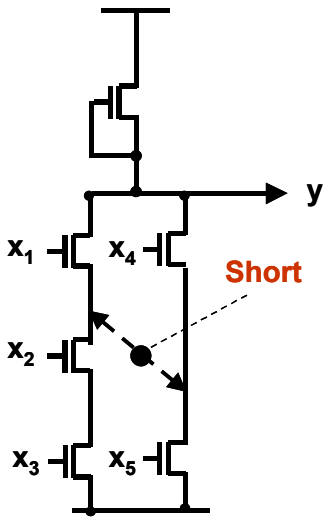

Fig.1. Transistor circuit with a short

To find W^d for a given defect d we have to create the corresponding logic expression for the faulty function f^d either by logical reasoning, or by carrying out directly defect simulation, or by carrying out real experiments to learn the physical behavior of different defects.

Let us have a transistor circuit in Fig.1 which implements the function $y = x_1x_2x_3 \vee x_4x_5$. A short defect as shown in Fig.1 changes the function of the circuit as follows: $y^d = (x_1 \vee x_4)(x_2x_3 \vee x_5)$. Using the defect variable

d for the short, we can create a generic differential equation for this defect and simplify the created expression as follows:

$$\frac{\partial y^*}{\partial d} = \frac{\partial[(x_1x_2x_3 \vee x_4x_5)\bar{d} \vee (x_1 \vee x_4)(x_2x_3 \vee x_5)d]}{\partial d} =$$
$$= \bar{x}_1\bar{x}_2x_4x_5 \vee x_1\bar{x}_3\bar{x}_4x_5 \vee \bar{x}_1x_2x_3\bar{x}_4\bar{x}_5 = 1$$

From the equation three possible solutions follow: T = {10x01, 1x001, 01110}. Each of them can be used as a test pattern for the given short. On this contra-example, it is easy to show the inadequacy of the stuck-at fault (SAF) model for testing the transistor level faults. For example, the set of five test patterns 1110x, 0xx11, 01101, 10110, 11010 which test all the stuck-at faults in the gate-level circuit does not include any of the possible test solutions for detecting the short from the set T.

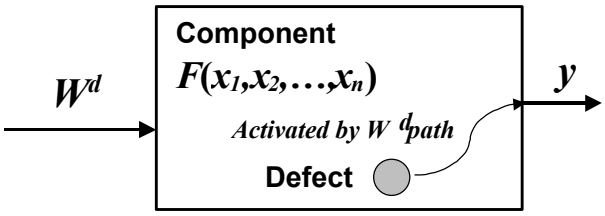

Fig. 2. Modelling a physical defect by a logic constraint

The described method represents a general approach to map an arbitrary physical defect onto a higher (in this case, logic) level. By the described approach, an arbitrary physical defect in a component can be represented as a logical constraint $W^d = 1$ to be fulfilled for activating the defect (Fig.2).

The event of erroneous value on the output y of a functional component can be described as $dy = 1$, where dy means Boolean differential. A functional fault representing a defect d can be described as a couple (dy, W^d). At the presence of a physical level defect d, we will have an higher level erroneous signal $dy = 1$ iff the condition $W^d = 1$ is fulfilled.

3. HIERARCHICAL APPROACH TO DIGITAL TEST

The method of defining faults by logic constraints W^d allows us to unify the diagnostic modelling of components of a circuit (or system) without going into structural details of components and into the diagnostic simulation of interconnection network of components. In both cases, a condition W^d describes how a lower level fault d (either a defect in a component or a defect in a network) should be activated at a higher level to a given node in a circuit (or system). The conditions W^d can be used both in fault simulation and in test generation.

Consider a node k in a circuit (Fig.7) as the output of a module M_k, and represented by a variable x_k. Associate with the node k a set of faults $R_k = R^F_k \cup R^S_k$ where R^F_k is the subset of faults in the module M_k, and R^S_k is a subset of structural faults (defects) in the "network neighborhood" of M_k. Denote by W^d the condition when the fault $d \in R_k$ will change the value of x_k. Denote by W^F_k the set of conditions W^d activating the defects $d \in R^F_k$ and by W^S_k the set of conditions W^d activating the defects $d \in R^S_k$.

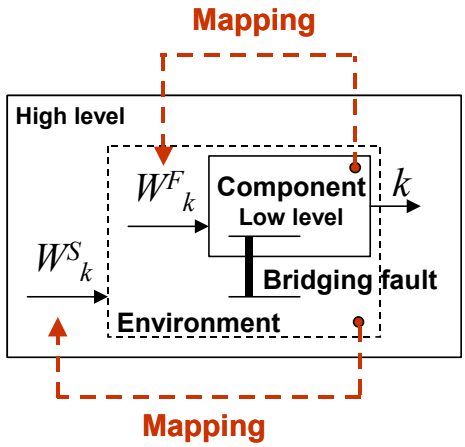

Fig. 3. Mapping faults from lower level to higher level

By using the sets of conditions W^F_k and W^S_k we can set up a mapping of faults from a lower level to a higher level for test generation purposes, and also in opposite direction, from a higher level to a lower level for fault simulation or fault diagnosis purposes.

In test generation, to map a lower level fault $d \in R_k$ to the higher level variable x_k, a solution of the equation $W^d = 1$ is needed. In other words, if the condition $W^d = 1$ is fulfilled then the presence of the defect $d \in R_k$ will change the value of the variable x_k.

In fault simulation (or in fault diagnosis) an erroneous value of x_k (denoted by a Boolean differential $dx_k = 1$) can be formally explained by implication

$$dx_k \rightarrow d_1W^{d1} \vee d_2W^{d2} \vee \ldots \vee d_nW^{dn} \qquad (3)$$

where for $j = 1,2,\ldots n$: $d_j \in R_k$. To the higher level event $dx_k = 1$, we set into correspondence a lower level event d_j if the condition $W^{dj} = 1$ is fulfilled.

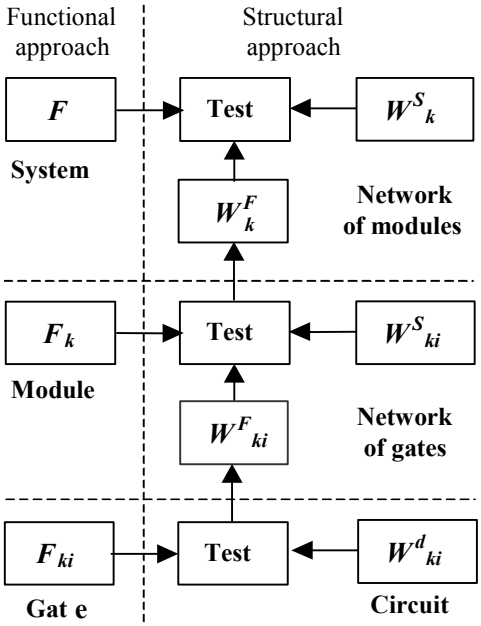

Fig.4. Hierarchical approach to testing

For hierarchical testing purposes we should construct for each module M_k of the circuit a list of faults R_k with logical conditions W^d for each fault $d \in R_k$. The set of conditions W^{F}_k for the functional faults $d \in R^{F}_k$ of the module can be found by low level test generation for the defects in the module. The set of conditions W^{S}_k for the structural faults $d \in R^{S}_k$ in the environment of the module can be found by Boolean differential analysis of generic fault-free/faulty functions as explained in Sections 2,3 and 4.

In Fig.4, a hierarchical test concept based on parametric fault modeling and functional fault model for a 3-level system is illustrated. In the functional approach, only the information about the functional behaviour is used. In the structural approach, tests are targeted to detect the faults in the networked components and in the network interconnections.

Consider a task of defect oriented fault simulation in a system which is represented at three levels: register transfer, gate and defect levels.

Formally, if Y is the system variable representing an observable point (a register) of the system, y_M is an output variable of a logic level module and y_G is the output of a logic gate with a physical defect d, then the condition to detect the defect d on the observable test point Y of the system is

$$W = \partial Y/\partial y_M \wedge \partial y_M/\partial y_G \wedge W^d = 1,$$

where $\partial Y/\partial y_M$ means the fault propagation condition calculated by high-level modeling, $\partial y_M/\partial y_G$ is the fault propagation condition (Boolean derivative) calculated by gate-level modeling, and W^d is the functional fault condition calculated from (2) by the gate preanalysis.

4. MODELLING DIGITAL SYSTEMS BY DECISION DIAGRAMS

A decision diagram [7] is defined as a non-cyclic directed graph $G = (M,\Gamma,X)$ with a set of nodes M, a set of variables X, and a relation Γ in M. The nodes $m \in M$ are labelled by variables $x(m) \in X$ (constants or algebraic expressions of $x \in X$). For each value from a set of predefined values of a non-terminal node variable $x(m)$, there exists a corresponding output branch from the node m into a successor node $m' \in \Gamma(m)$. Consider a situation where all variables are fixed to some value. By these values, for each non-terminal node m a certain output branch is chosen, which is connected to a successor node. Let us call these connections between nodes - *activated branches*, and the chains of them - *activated paths.* For each combination of values of variables, there exists always a *full activated path* from the root node to a terminal node. This relation describes a mapping from a Cartesian product of the sets of values for variables in all nodes to the joint set of values for variables (or expressions) in terminal nodes. Therefore, by DDs it is possible to represent arbitrary digital functions $Y=F(X_Y)$, where Y is the variable whose value will be calculated by the DD and X_Y is the vector of all variables in nodes of the DD.

Using DDs we can efficiently carry out the following tasks of digital test: test pattern simulation, fault simulation, and test generation. In the following we describe how these tasks are solved with DDs on different system representation levels.

A. Binary decision diagrams and gate-level circuits

A BDD that represents a Boolean function is a directed noncyclic graph with single root node, where all nonterminal nodes are labelled by (inverted or not 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 represented the graph. For simplification, the

Unlike the traditional BDDs [4-6], structurally synthesized BDDs [7,8] support structural representation of gate-level networks in terms of signal paths. By superposition procedure described in [7,8], we create SSBDDs where one-to-one correspondence between nodes and signal paths in tree-like subcircuits exists. The whole digital circuit can be represented as a network of tree-like subcircuits (macros), each of them represented by a SSBDD. Using SSBDDs, it is possible to ascend from the gate-level descriptions of circuits to higher macro level descriptions without loosing accuracy of representing gate-level signal paths.

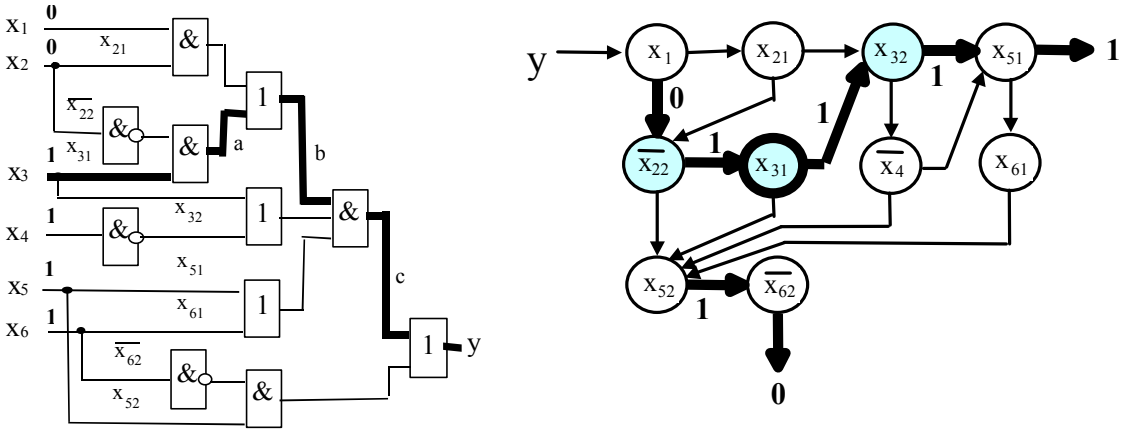

Fig.5. Gate level circuit and SSBDD

As an example, Fig.5. shows a representation of a tree-like combinational macro by a SSBDD. By convention, the right-hand branch corresponds to 1 and the lower-hand branch to 0. In test pattern simulation, a path is traced in the graph, based on the values of input variables until a terminal node is reached. The value of the variable x in the terminal node is taken as the result of simulation, and this value will be assigned to the output variable y. In this example, the result of simulating the vector $x_1\, x_2\, x_3\, x_4\, x_5\, x_6 = 001111$ is $y = 1$ (bold arrows mark the path activated by the input pattern).

Fault simulation is carried out by the following analytical procedure. Suppose that a test pattern activates a full path up to a terminal node with a variable x_{t1}. For all the nodes on the full activated path, it will be cheked by simulation if the change of the value of the node variable x will lead to another terminal node with a variable x_{t2}. If yes, and if $x_{t1} \neq x_{t2}$ then the fault in x is detected by the given test pattern, otherwise not. For example, in Fig. 5, the following faults in the full activated path are detected by the given test pattern: $\neg x_{22} \equiv 0$, $x_{31} \equiv 0$, and $x_{32} \equiv 0$. Note that, since the nodes in SSBDDs represent the signal paths, it is possible to fault simulate with SSBDD by one simulation run a whole group of faults. For example, since the node x_{31} in SSBDD represents the path $L=(x_{31},a,b,c,y)$ in the circuit, it results from detecting the fault $x_{31} \equiv 0$ in SSBDD that all the faults $x_{31} \equiv 0$, $a \equiv 0$, $b \equiv 0$, $c \equiv 0$, $y \equiv 0$ on the path L are detected.

In test generation we first, choose a target node x to be tested. Then we activate a path to x from the root node, and two paths from the successor nodes of x to different terminal nodes x_{t1} and x_{t2}, so that $x_{t1} \neq x_{t2}$. For example, to test the node x_{31}, we activate a path to x_{31} by choosing the values $x_1=0$, and $x_2=0$. Then we activate by assigning $x_3=1$ a path from x_{31} to the terminal node $x_{t1} = x_{51}$ and assign $x_5=1$. Finally, we have to activate from x_{31} a path to another terminal node x_{t2} with purpose to solve the inequality $x_{51} \neq x_{t2}$. The first trial would be to choose x_{52} as such a terminal node x_{t2}. However, at this choice the inequality $x_{t1} \neq x_{t2}$ cannot be solved. The earlier assigned value $x_5=1$ directs us to the next node where we find the value $x_{62}=1$ to solve the condition $x_{51} \neq \neg x_{62}$. The assigned values $x_1\, x_2\, x_3\, x_4\, x_5\, x_6 = 001u11$ (where u means *don't care*) form the test pattern which detects the fault at the node x_{31}. Since $x_3 = 1$, the test pattern detects the fault $x_{31} \equiv 0$.

B. High-level decision diagrams and register-transfer level circuits

Depending on the class of the system (or its representation level), we may have various DDs, where nodes have different interpretations and relationships to the system structure. In register transfer level (RTL) descriptions, we usually decompose digital system into control and data parts. State and output variables of the control part serve as addresses and control words, and the variables in the data part serve as data words. High-level data word variables describe RTL functions in data parts. When using DDs for describing complex digital systems, we have to first, represent the system by a suitable set of interconnected components (combinational or sequential subcircuits). Then, we have to describe these components by their corresponding functions which can be represented by DDs.

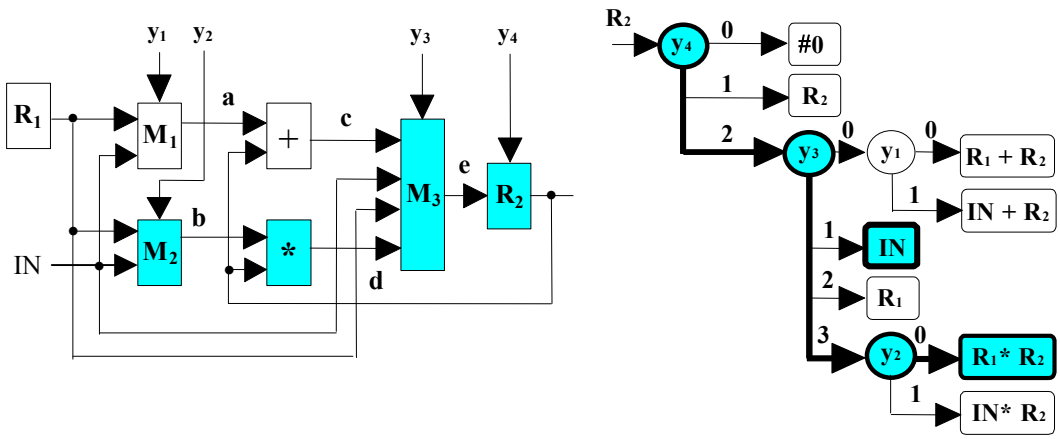

Fig.6. Register-Transfer Level level Data-Path System

As an example, in Fig.6 a RTL data-path and his compressed DD is presented. The DD is created by superposition of elementary DDs of components of the circuit. The word variables R_1, R_2 and R_3 represent registers, the integer variables y_1, y_2 , y_3, and y_4 represent the control signals. M_1, M_2 and M_3 are multiplexers, and the functions R_1+R_2 and R_1*R_2 represent the adder and multiplier, correspondingly. The whole DD describes the behaviour of the input logic of the register R_2.

In test pattern simulation, a path is traced in the graph, guided by the values of input variables until a terminal node is reached, similarly as in the case of BDDs. In this example, the result of simulating the vector y_1, y_2, y_3, y_4, R_1, R_2, IN = 0,0,3,2,10,6,12 is $R_2 = R_1*R_2 = 60$ (bold arrows mark the path activated by the control pattern). Instead of simulating by a traditional approach all the components in the circuit, on the DD only 3 control variables are visited during simulation, and only a single data manipulation $R_2 = R_1*R_2$ is carried out.

Each node in DD represents a subcircuit of the system. For example, the nodes y_1, y_2, y_3, y_4 represent multiplexers and decoders, the nodes R_1, R_2, IN, and other terminal nodes represent, correspondingly registers, input bus, and data manipulation subcircuits. To test a node means to test the corresponding subcircuit.

Tests for the terminal nodes are generated hierarchically: the test program C (sequence of control words) is generated on high-level, the test data D for activating low level faults are generated on low level. For example, to test the node R_1*R_2 we activate a full path to this node by assigning: y_4=2, y_3=3, y_2=0 (bold arrows in Fig.6). The generated control word $C = y_1, y_2, y_3, y_4$ = u,0,3,2 (u – means *undefined* or *don't care*) will select the microoperation R_1*R_2 under test. The data D needed for testing the microoperation R_1*R_2 is generated on gate-level (using SSBDDs) to be able to process all the low level faults of the given implementation. The whole test will be carried out as the following sequence: 1) Load R_2 from D, 2) Load R_3 from D, 3) Apply C, 4) Read R_2. This sequence is carried cyclically for all the data D.

Test generation for the internal (control) nodes in DD can be carried only on the high-level using the exhaustive fault model for nodes. First, we select a target node x for testing. Then we activate a path to x, and the paths from all of its successor nodes x_1, x_2,…, x_n, to not coinciding terminal nodes x_{t1}, x_{t2},…, x_{tn} so that $x_{t1} \neq x_{t2} \neq \ldots \neq x_{tn}$. Let us call T_k as a test which activates a full path to the terminal node x_{tk}. The whole test consists of n cycles $T = \{T_k \mid k = 1,2,\ldots, n\}$, where each cycle is carried out as the following sequence: 1) Load all the registers with data to satisfy the condition $x_{t1} \neq$

$x_{t2} \neq \ldots \neq x_{tm}$, 2) carry out the test T_k, 3) observe the result. For example, to test the node y_3 in Fig.6, we activate by assigning y_4=2 the path from y_4 to y_3, then by assigning y_1=0 the path from y_3 to R_1+R_2, and by assigning y_2=0 the path from y_3 to R_1*R_2. The paths from y_3 to *IN*, and to R_1 are always activated. The generated control word is $C = y_1, y_2, y_3, y_4 = 0,0,k,2$. The data for testing y_3 are chosen by solving the inequality $R_1+R_2 \neq IN \neq R_1 \neq R_1*R_2$. The whole test consists of 4 cycles: $k = 1,2,3,4$.

C. Vector decision diagrams for representing digital systems

Consider a digital system in Fig. 7. The system consists of control and data parts. The FSM of the control part is given by the output function $y = \lambda\,(q', x)$ and the next-state function $q = \delta\,(q', x)$, where y is an integer output vector variable, which represents a microinstruction with four control fields $y = (y_M, y_z, y_{z,1}, y_{z,2})$, $x = (x_A, x_C)$ is a Boolean input vector variable, and q is the integer state variable. The value j of the state variable corresponds to the state s_j of the FSM. The data path consists of the memory block M with three registers A, B, C together with the addressing block *ADR*, which can be represented by three DDs: $A = G_A\,(y_M, z)$, $B = G_B\,(y_M, z)$, $C = G_C\,(y_M, z)$; of the data manipulation block CC where $z = G_z\,(y_z, z_1, z_2)$; and of two multiplexers $z_1 = G_{z,1}\,(y_{z,1}, M)$ and $z_2 = G_{z,2}\,(y_{z,2}, M)$. The block COND performs the calculation of the condition function $x = G_x\,(A, C)$. Hence, the component level model of the system can be represented by the following set of DDs: G_q, G_y, G_A, G_B, G_C, G_z, $G_{z,1}$, $G_{z,2}$, G_x. In each step of the cycle-based simulation we have to carry out simulation in each of these graphs.

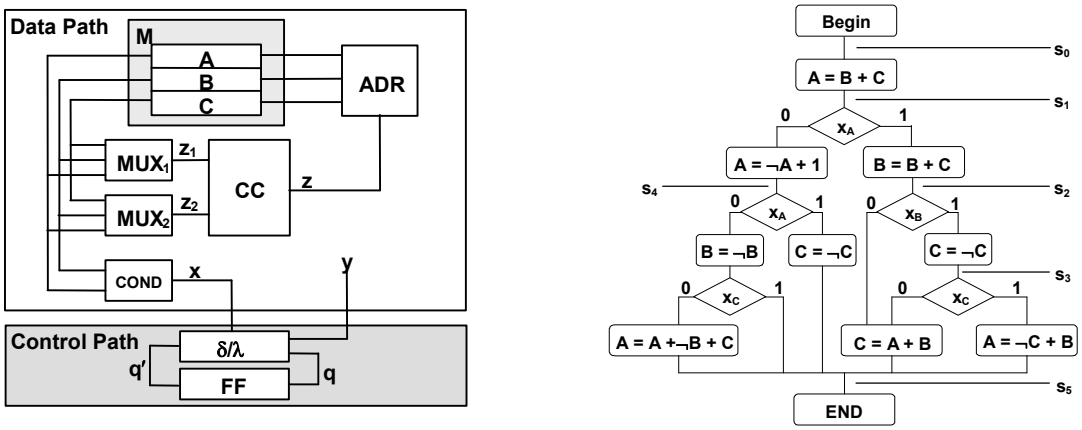

Fig.7. A digital system as a structure and a microprogram

In Fig. 8, another model of the system is presented, which consists of only 4 DDs: G'_q, G'_A, G'_B, G'_C. Reducing the number of graphs by superposition allows to increase the simulation speed. The graphs are constructed by *superposition* of DDs [8]. For example, the DD for register A can be constructed as follows:

$$A = G_A\,(y_M, z) = G_A\,(y_M, G_z\,(y_z, z_1, z_2)) = G_A\,(y_M, G_z\,(y_z, G_{z,1}\,(y_{z,1}, M), f_4\,(y_{z,2}, M))) =$$

$$= G_A\,(y_M, y_z, y_{z,1}, y_{z,2}, M) = G_A\,(y, M) = G_A\,(G_y\,(q', x), M) = G'_A\,(q', A, B, C).$$

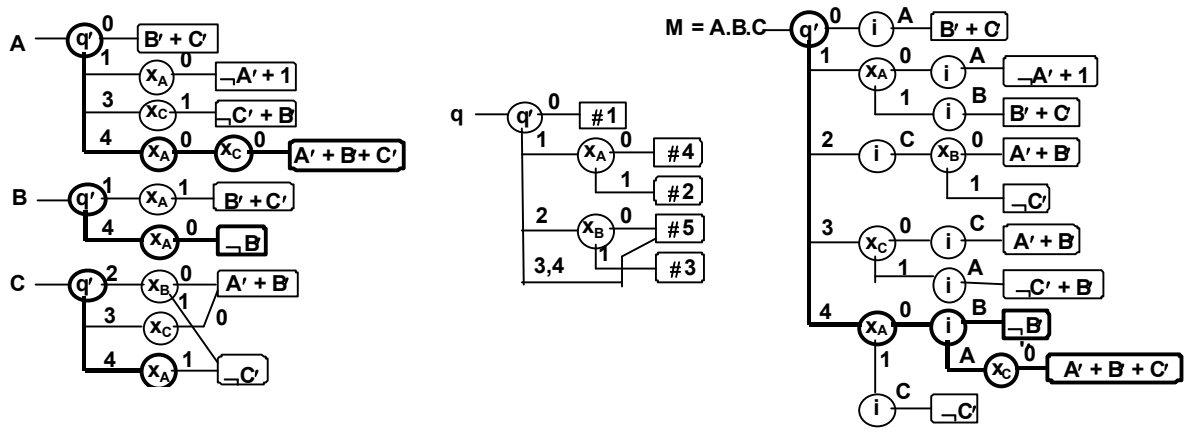

Fig.8. A digital system as a structure and a microprogram

Consider now a method of joining a set of DDs into a single vector DD. In Fig.8 the DDs G_A, G_B and G_C are joined into a single vector DD (VDD) $M = A.B.C = G_M\ (q', A, B, C, i)$ which produces a new concise model of the system in Fig. 7. For calculating and assigning new values to different components of the vector variable M, we introduce a new type of a node in VDD called *addressing node* labeled by an addressing variable i. The vector decision diagrams offer the capability to efficiently represent the array variables (corresponding to register blocks and memories) for calculating and updating their values. VDDs are particularly efficient for representing functional memories with complex input logic – with shared and dedicated parts for different memory locations. In general case, all the registers of the data path can be combined in the model as a single memory block.

Using VDDs allows significally to increase the speed of simulation. For example, in G_M in Fig. 8 for the input vector $q' = 4,\ x_A = 0,\ x_C = 0,$ the nodes q' and x_A, are traversed for calculating both new values of A and B only once, whereas in case of separate DDs they should be traversed for all the graphs: separately for A, B and C.

CONCLUSIONS

Two main trends can be observed today in the field of digital test: defect-orientation to increase the quality of testing, and high-level modelling to reduce the complexity problems of diagnostic analysis. However, on the other hand, counting defects increases the complexity, and high-level modelling reduces the accuracy. Hierarchical approaches can solve this antagonism. Decision diagrams discussed in the paper contribute as a good tool for hierarchical modelling and diagnostic analysis of digital systems. The new innovative forms of DDs, high-level and vector decision diagrams have been discussed. From simulation point of view, they provide a compact and efficient representation of a digital system. High-level DDs is a new model, and there are a lot of possibilities for further research, for additional improvements and optimization.

ACKNOWLEDGEMENTS

This work has been supported by EU V Framework projects REASON and EVIKINGS, as well as by the Estonian Science Foundation grants 4300 and 5649.

REFERENCES

[1] J.Lee and J.H.Patel. ARTEST: “An Architectural Level Test Generator for Data Path Faults and Control Faults”. *Proc. Int. Test Conf.* Oct. 1991, pp. 729-738.

[2] S.R.Rao, B.Y.Pan, J.R.Armstrong: “Hierarchical Test Generation for VHDL Behavioral Models”. *Proc. EDAC*, Feb. 1993, pp. 175-183.

[3] E.M.Rudnick, R.Vietti, A.Ellis, F.Corno, P.Prinetto, M.Sonza Reorda: "Fast sequential circuit test generation using high-level and gate-level techniques", *Proc. of DATE*, 1998.

[4] R.E.Bryant: “Graph-based algorithms for Boolean function manipulation”. *IEEE Trans. on Computers,* Vol.C-35, No8, 1986, pp.667-690.

[5] S. Minato: “BDDs and Applications for VLSI CAD”. Kluwer Academic Publishers, 1996, 141 p.

[6] R.Drechsler, B.Becker: “BDDs. Theory and Implementation”. *Kluwer Academic Publishers,* 1998, 200 p.

[7] R.Ubar: “Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams”. *OPA (Overseas Publishers Assotiation) N.V. Gordon and Breach Publishers, Multiple Valued Logic,* Vol.4, 1998, pp. 141-157.

[8] R.Ubar: “Test Synthesis with Alternative Graphs”. *IEEE Design&Test of Computers*, Spring 1996,pp.48-57.

[9] J.Raik, R.Ubar: “Sequential Circuit Test Generation Using Decision Diagram Models”. *IEEE Proc. of Design Automation and Test in Europe*, Munich, March 9-12, 1999, pp. 736-740.

[10] R.Ubar, A.Morawiec, J.Raik. Back-Tracing and Event-Driven Techniques in High-Level Simulation with Decision Diagrams. Proc. of the IEEE ISCAS’2000 Conf., Geneva, May 28-31, 2000, Vol. 1, pp. 208-211

[11] J.M. Soden, C.F. Hawkins: “Quality Testing Requires Quality Thinking”. *Proc. ITC,* 1993, pp. 596.

[12] L.M. Huisman: “Fault Coverage and Yield Predictions: Do We Need More than 100% Coverage”? *Proc. of European Test Conference,* 1993, pp. 180-187.

[13] W.Maly, J.P.Shen, and J.Ferguson: “System. Characterization of Physical Defects for Fault Analysis of MOS IC Cells”. *Proc. Int. Test Conf.,* 1984, 390-399.

[14] P.Nigh and W.Maly: “Layout - Driven Test Generation”. *Proc. ICCAD,* 1989, 154-157.

[15] M.Jacomet and W.Guggenbuhl: “Layout-Dependent Fault Analysis and Test Synthesis for CMOS Circuits”. *IEEE Trans. on CAD,* 1993, **12,** 888-899.