DECISION DIAGRAMS AND DIGITAL TEST

Raimund Ubar

Tallinn University of Technology, Tallinn, Estonia

INVITED PAPER
MIDEM 2005 CONFERENCE
14.09.2005 - 16.09.20045, Bled, Slovenia

Key words: testing, testing of digital systems, decision diagrams, hierarchical modeling, high level decision diagrams, vector descision diagrams

Abstract: *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. Finally, two examples of hierarchical test generation tools based on DDs together with corresponding experimental results are given.*

Odločitveni diagrami in digitalno testiranje

Kjučne besede: testiranje, testiranje digitalnih sistemov, odločitveni diagrami, hierarhično modeliranje, odločitveni diagrami na višjem nivoju, vektorski odločitveni diagrami

Izvleček: Eno najpomembnejših današnjih vprašanj pri testiranju digitalnih sistemov je, kako izboljšati zanesljivost testiranja ob stalnem naraščanju kompleksnosti sistemov ? Opažamo dvae glavni smeri : eno, ki je usmerjena k odkrivanju napak in drugo, ki obsega modeliranje na višjem nivoju, kar zmanjša zapletenost diagnostične analize. Obe smeri lahko združimo v hierarhični pristop. Odločitveni diagrami (DD) služijo kot dobro orodje za hierarhično modeliranje in diagnostično analizo digitalnih sistemov. Tradicionalni binarni odločitveni diagrami (BDD) so dobro poznani pri delu z logičnimi nivoji. V prispevku obravnavamo nove posplošitve BDD v obliki DD na višjem nivoju in vektorske DD kot učinkovita orodja za tvorbo testov in simulacijo napak pri kompleksnih digitalnih sistemih. Na koncu podamo dva primera orodja za tvorbo hierarhičnih testov na osnovi DD skupaj z ustreznimi eksperimentalnimi rezultati

1. Introduction

Test generation for digital systems encompasses three activities: selecting a description method, developing a fault model and generating tests to detect the faults covered by the fault model. The efficiency of test generation (quality, speed) is highly depending on the description method and fault models which have been chosen.

As the complexity of digital systems continues to increase, the gate level test generation methods have become obsolete. Other approaches based mainly on higher level functional and behavioral methods are gaining more popularity /1-3/. However, the trend towards higher level modelling moves us even more away from the real life of defects and, hence, from accuracy of testing. To handle adequately defects in deep-submicron technologies, new fault models and defect-oriented test methods should be used. On the other hand, the defect-orientation is increasing even more the complexity. To get out from the deadlock, the two opposite trends - high-level modelling and defect-orientation - should be combined into hierarchical approaches. The advantage of hierarchical approaches compared to high-level functional modelling lies in the possibility of constructing test plans on higher levels, and modelling faults on more detailed lower levels.

The drawback of traditional multi-level and hierarchical approaches to digital test lies in the need of different dedicated languages and models for different levels. Uniform methods for hierarchical diagnostic modelling of digital systems can be developed by using Decision Diagrams (DD) /4-9/. Binary DDs (BDD) have found already very broad applications in design and test on the logic level /4-5/. A special class of BDDs, Structurally Synthesized BDDs (SSBDD) can be used to represent gate-level structural faults directly in the graph model /6,7/. Recent research has shown that generalization of BDDs for higher levels provides a uniform model for both gate and RT level or even behavioral level test generation /8,9/.

The disadvantage of the traditional hierarchical test approaches is the use of gate-level stuck-at fault (SAF) model. It has been shown that high SAF coverage cannot quarantee, high quality of testing /10/. 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 but usually, they have been ignored in engineering practice. In earlier works on layout-based test techniques /11,12/ 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.

In this paper, we present, first, in Section 2 a method for mapping faults from lower levels to higher levels. For this purpose the concept of functional fault model is used. Thereafter, for hierarchical diagnostic modelling of digital systems, DDs are presented. In Section 3 SSBDDs are described for logic level test generation, and in Section 4 the use of higher level DDs for test generation is discussed. Some experimental data are presented in Section 5 to illustrate the efficiency of the described hierarchical approach. Section 6 concludes the paper.

2. Functional fault model in hierarchical test

Consider a Boolean function $y = f(x_1, x_2, ..., x_n)$ implemented by an embedded component C in a digital 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 faulty function $y = f^d(x_1, x_2, ..., x_n)$. 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 the defect variable d, which describes the behavior of the component simultaneously for both possible fault-free and faulty cases. 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) allows us to use the constraints $W^d = 1$, either in defect-oriented fault simulation to check if the condition (2) is fulfilled (i.e. if the defect d is tested by the given pattern), or in defect-oriented test generation to solve the equation (2) for testing the defect d. The conditions W^d allow to map physical defects to logic level. The constraint $W^d = 1$ defines how a lower level fault d should be activated at a higher level to a given node y.

Table 1: Activating conditions for different defects

No	Defect	Conditions W^d
1	SAF $x_k \equiv 0$	$x_k = 1$
2	SAF $x_k \equiv 1$	$x_k = 0$
3	Short between x_k and x_l	$x_k = 1, x_l = 0$
4	Exchange of lines x_k and x_l	$x_k = 1, x_l = 0$, or $x_k = 0, x_l = 1$
5	Delay fault on the line x_k	$x_k = 1, x'_k = 0$, or $x_k = 0, x'_k = 1$

Table 2: Library defect table for a complex gate AND2,2/NOR2

i	Fault d_i	Erroneous function f^{di}	Input patterns t_j															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	B/C	not((B*C)*(A+D))				1								1	1	1		
2	B/D	not((B*D)*(A+C))				1								1	1		1	
3	B/N9	B*(not(A))	1	1	1					1	1	1	1					
4	B/Q	B*(not(C*D))	1	1	1						1	1	1		1	1	1	
5	B/VDD	not(A+(C*D))									1	1	1					
6	B/VSS	not(C*D)													1	1	1	
7	A/C	not((A*C)*(B+D))				1				1					1	1		
8	A/D	not((A*D)*(B+C))				1				1					1		1	
9	A/N9	A*(not(B))	1	1	1		1	1	1					1				
10	A/Q	A*(not(C*D))	1	1	1		1	1	1						1	1	1	
11	A/VDD	not(B+(C*D))					1	1	1									
12	C/N9	not(A+B+D)+(C*(not((A*B)+D)))		1			1	1			1	1						
13	C/Q	C*(not(A*B))	1	1		1	1	1		1	1	1		1				
14	C/VSS	not(A*B)				1				1				1				
15	D/N9	not(A+B+C)+(D*(not((A*B)+C)))			1		1		1		1		1					
16	D/Q	D*(not(A*B))	1		1	1	1		1	1	1		1	1				
17	N9/Q	not((A*B)+(B*C*D)+(A*C*D))				1												
18	N9/VDD	not((C*D)+(A*B*D)+(A*B*C))													1			
19	Q/VDD	SA1 at Q				1				1				1	1	1	1	1
20	Q/VSS	SA0 at Q	1	1	1		1	1	1		1	1	1					

Some examples of the conditions W^d for different type of defects (where SAF is a particular extreme case) are given in Table 1 (here x_k is the observable variable, and x'_k is the variable observed at the previous time moment).

The event of erroneous value of y 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 the physical level defect d, we will have an higher level erroneous signal $dy = 1$ if the condition $W^d = 1$ is fulfilled.

The functional fault model (dy, W^d) allows to use for test generation for any physical defect d traditional stuck-at fault test generators. To generate a test for a defect d, a test pattern should be generated for the stuck-at fault $y \equiv (1 \oplus y(W^d))$ at the additional condition $W^d = 1$. Here $y(W^d)$ is the expected value of y determined by the condition $W^d = 1$.

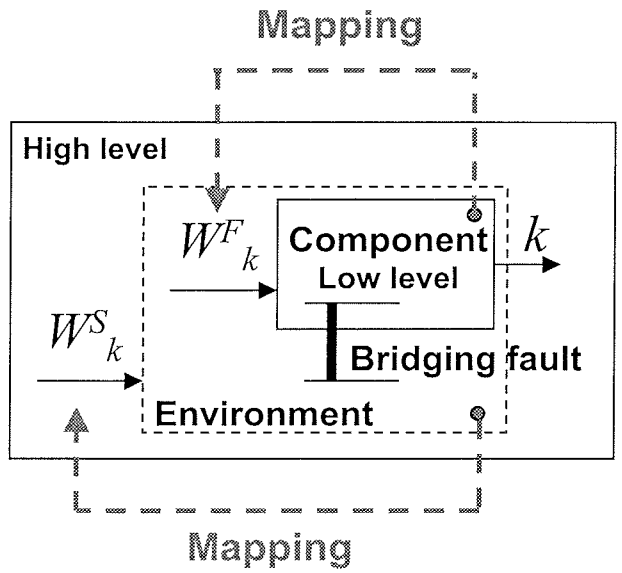

Figure 1: Mapping faults from lower level to higher level

In the described approach we have to characterize all possible defects in all library cells, and represent the results as defect tables or as optimized sets of defect activating conditions $W^F = \{W^{d,i}\}$. The defect characterization may be computationally expensive, but it is performed only once for every library cell. An example of the fault table for the complex gate AND2,2/NOR2 with a function y = ¬(AB Ú CD) is presented in Table 2 /13/.

The defect lists W^F_k of library components C_k embedded in the circuit can be extended by additional physical defect lists W^S_k in the close network environment of the component C_k to take into account also the wrong behaviour of C_k influenced by the outside environment (bridging faults, crosstalkings etc.). For these defects additional characterization should be carried out by a similar way as for the library cells.

3. Diagnostic modelling of digital systems by BDDs

Decision Diagrams (DD) can serve as a basis for a uniform approach to test generation for mixed-level representations of systems, similarly as we use the Boolean algebra for the plain logic level. In the following it is shown how the traditional logic level test methods can be implemented on Binary Decision Diagrams (BDD) /6,7,14/ as a special class of DDs, and then we generalize the procedures developed for BDDs for a general class of DDs /6,16,17/ to handle the test generation problems at higher levels of systems.

Structurally synthesized BDDs. In 1959 C.Y.Lee introduced a method for representing digital circuits by Binary Decision Programs /18/. In 1976 /14/ and in 1977 /15/ independently the BDDs were introduced for test generation purposes. Today the theory of BDDs is developing quickly /4,5,19/.

In /7,14/ structurally synthesized BDDs (SSBDD) as a special class of BDDs was introduced to represent the topology of gate-level circuits in terms of signal paths. Unlike "traditional" BDDs /4,18/, SSBDDs directly support test generation for gate-level structural faults without explicitly representing the faults. The advantage of SSBDDs is that the library of D-cubes for components is not needed for structural path activization. That's why SSBDD based test generation procedures do not depend on whether the circuit

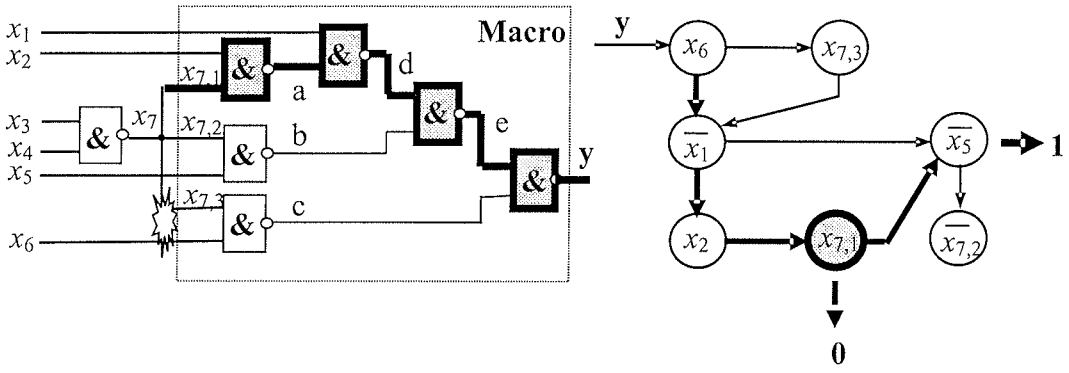

Figure 2: Combinational macro and his SSBDD

is represented on the gate level or on the higher macro-level whereas the macro means an arbitrary single-output subcircuit of the whole circuit. Moreover, the test generation procedures developed for SSBDDs can be easily generalized for higher level DDs to handle digital systems represented at higher levels /6,16,17/.

The BDD that represents a Boolean function is a directed noncyclic graph with a single root node, where all nonterminal nodes are labelled by Boolean variables (arguments of the function) and have always exactly two successor-nodes whereas the terminal nodes are labelled by constants 0 or 1. For all nonterminal nodes, a one-to-one correspondence exists between the values of the label variable of the node and the successors of the node. The correspondence is determined by the Boolean function to be represented by the graph.

Denote the variable which labels a node m in a BDD by $x(m)$. We say that a value of the node variable activates the node output edge. According to the value of $x(m)$, one of two output edges of m will be activated. If $x(m) = 1$ we say 1-edge is activated, or if $x(m) = 0$ we say 0-edge is activated. A path is activated if all the edges that form this path are activated. The BDD is activated to 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).

Definition 3.1. A BDD G_y with nodes labelled by variables $x_1, x_2, ..., x_n$, represents a Boolean function $y = f(X) = f(x_1, x_2, ..., x_n)$, if for each pattern of X, the BDD will be activated to the value which is equal to y for the same pattern.

Important property of SSBDDs. SSBDDs differently from traditional BDDs have the following property: each node m in a G_y which describes a tree-like subnetwork N_y of the gate-level circuit N, represents a signal path $l(m)$ in N_y. There is an one-to-one correspondence between the nodes m in a G_y and the paths $l(m)$ in the corresponding circuit N_y.

An example of a combinational circuit with a tree-like macro and SSBDD for the macro is presented in Figure 2. For simplicity, the values of variables on edges of the SSBDD are omitted (by convention, the 1-edge is always directed to the right, and the 0-edge is always directed downwards). Also, terminal nodes with constants 0 and 1 are omitted (leaving the SSBDD to the right corresponds always to $y = 1$, and down - to $y = 0$). Each node is marked by an input variable of the macro. A node with the label x_m in the SSBDD represents the signal path through the macro which begins with the input variable x_m. The node variable is inverted when the path consists of odd number of inverters, and not inverted when the number of inverters is even. For example, the node $x_{7,1}$ of SSBDD represents the signal path with even number of inverters starting with the line $x_{7,1}$ through the nodes a,d,e to the output y in the macro (the bold lines in the circuit). The node in the SSBDD has inverted variable since the corresponding path x_1,d,e,y consists of odd number of inverters. The fan-out node x_7 in the circuit has three branches, and each branch $x_{7,i}$ (i = 1,2,3) is the beginning of a path which is represented by the node $x_{7,i}$ in the SSBDD.

From the above described property of the SSBDD, automatic fault collapsing results. Assume a node m with label variable $x(m)$ represents a signal path $l(m)$ in a circuit. Suppose the path $l(m)$ goes through n gates. Then, instead of $2n$ faults of the path $l(m)$ in the circuit, only 2 faults related to the node variable $x(m)$ should be tested when using the SSBDD model.

Test generation with SSBDDs. Consider a combinational circuit as a network of gates, which is partitioned into interconnected tree-like subcircuits (macros). This is a new higher level (macro-level) representation of the same circuit. Each macro is represented by a SSBDD where each node corresponds to an input of the macro. In the tree-like subcircuits only the stuck-at faults at inputs should be tested. This corresponds to testing all the nodes in each SSBDD. Test generation for a node m in SSBDD, which represents a function $y = f(X)$ of a macro, is carried out by the following procedure /19,23/.

Algorithm 1.

1) A path l_m from the root node of SSBDD to the node m is activated.
2) Two paths $l_{m,e}$ consistent with l_m, where $e \in \{0,1\}$, from the neighbors m^e of m to the corresponding terminal nodes $m^{T,e}$ are activated.
3) For generating a test for a particular stuck-at-e fault $x(m) \equiv e$, $e \in \{0,1\}$, the opposite assignment for $x(m)$ is needed: $x(m) = \bar{e}$.
4) All the values assigned to node variables build the local test pattern $T(X,y)$ (input pattern of the macro) for testing the node m in G_y.

The paths in the SSBDD activated by *Algorithm 1* are illustrated in Figure 3.

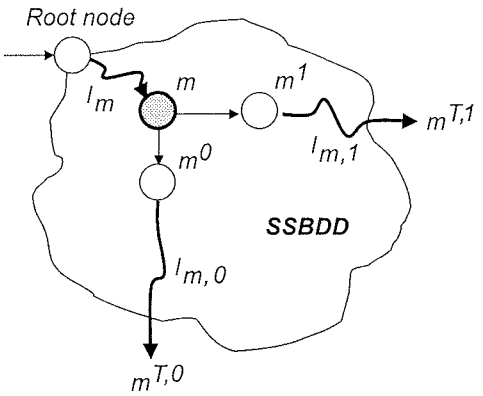

Figure 3: Test generation for the node m *with SSBDD*

To create the final test pattern in terms of primary inputs of the circuit (network of macros) for the given fault in an

embedded macro, fault propagation and line justification through the network of macros are needed. The fault propagation through a macro from the input x to its output y is carried out similarly to the test generation for the node m labelled by x in the corresponding SSBDD G_y as explained in *Algorithm 1*. Line justification for the task $y = e$ is carried out by activating a path in the graph G_y from the root node to the terminal node $m^{T,e}$.

Example 1. Consider test generation for the bridging defect between lines x_6 and x_7 of the circuit in Figure 2. This defect can be described by a functional fault model (dx_7, $\overline{x_6}x_7 = 1$), which corresponds to the AND-type bridging model. To generate a test for the short we can generate a test for the stuck-at-1 fault ($x_{7,1} \equiv 1$) of the internal line $x_{7,1}$ (input of the macro) in the circuit at aditional conditions $x_6 = 0$, $x_7 = 1$. Using SSBDD we have to generate by *Algorithm 1* a test for the node $x_{7,1}$ in the SSBDD in Figure 2 at the preconditions $x_6 = 0$, $x_7 = 1$. Activating the path l_m through the nodes x_6, $\overline{x_1}$, and x_2 gives new additional assignments $x_1 = 1$, and $x_2 = 1$. Activating the path $l_{m,1}$ through the node $\overline{x_5}$ gives $x_5 = 0$. The path $l_{m,0}$ is activated "automatically", since the 0-edge from the node $x_{7,1}$ is connected directly to the terminal node $m^{T,0}$. The paths, activated by the generated test pattern $x_1x_2x_5x_6x_7 = 11001$, are shown by bold lines in Figure 2.

4. Using high-level DDs for diagnostic modelling

Test generation and fault simulation methods developed for SSBDDs have the advantage compared to other logic level methods that they can be easily generalized to handle the test generation and fault simulation problems at higher system levels /6,16,17/.

In general case (beyond the Boolean algebra and BDDs) a decision diagram can be 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 /6/. 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 edge 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 edge is chosen, which is connected to a successor node. Let us call these connections between nodes - *activated edges*, and the chains of them - *activated paths*. For each combination of values of variables of X, 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 $Y=F(X)$, where Y is the variable whose value will be calculated by the DD and X is the vector of all variables in nodes of the DD.

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

As an example, in Figure 4 a RTL data-path and his compressed DD is presented /20/. The DD is created by superposition /6,21/ of elementary DDs for the components of the circuit. The word variables R_1, R_2 and R_3 represent

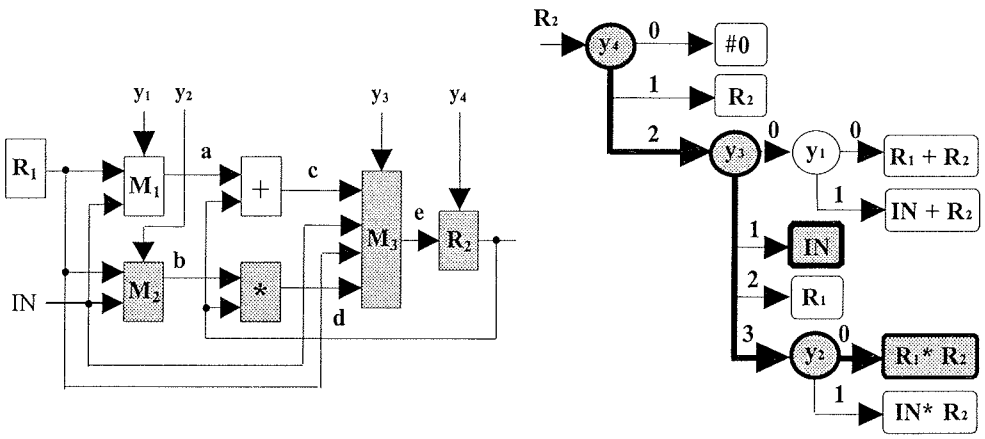

Figure 4: Register-transfer level data-path system

registers, *IN* representss the input bus, 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 SSBDDs. 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 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. To test a node means to test the corresponding subcircuit. The paths to be activated during test generation in high-level DDs are illustrated in Figure 5.

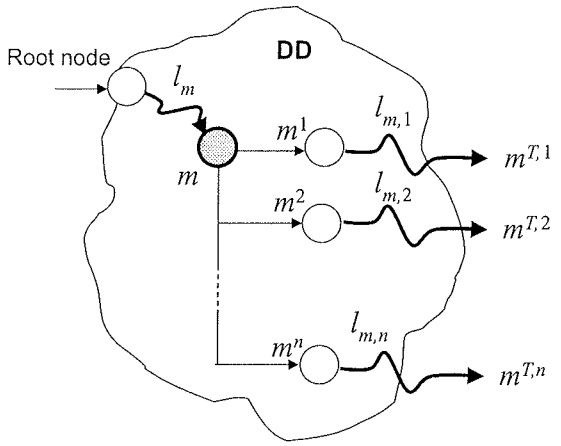

Figure 5: Test generation with high-level Decision Diagrams

We differentiate two testing types used for systems: *scanning test* (for testing terminal nodes in DDs, i.e. for testing the data path), and *conformity test* (for testing nonterminal nodes, i.e. for testing the control path).

Procedure 1. Scanning test. To generate a scanning test for a terminal node $m^{T,i}$ in the DD G_y, the path $l(m^{T,i})$ from the root node to $m^{T,i}$ is to be activated, and the test patterns for testing the function $z(m^{T,i})$ should be generated (according to the hierarchical approach, these patterns can be generated on the lower level representation of $z(m)$, and they can be regarded as a set of additional conditions W^F according to the functional fault model to map the faults from logic level to RT level).

Procedure 2. Conformity test. To generate a conformity test for a node m in G_y, the following paths are to be activated: 1) $l(m)$, and 2) for all the values of i = 1,2,...,n: $l(m,i)$, and the proper data are to be found by solving the inequality $x(m^{T,1}) \neq x(m^{T,2}) \neq ... \neq x(m^{T,n})$ where i = 1,2,...,n, and $x(m^{T,i})$ are the algebraic expressions of the terminal nodes.

To generate a scanning test for the node $R_1 * R_2$ of the DD in Figure 4, a path $l(R_1*R_2) = (y_4, y_3, y_2, R_1*R_2)$ is to be activated, and the data vetors (local test patterns) *DATA* = $(R_{1,1},R_{2,1}; R_{1,2},R_{2,2}; ... R_{1,m},R_{2,m})$ for testing the multiplier are to be generated at low level by any ATPG. The scanning test consists in cyclically run sequence: FOR all $(a,b) \in DATA$: /Load: $R_1 = a$; Load: $R_2 = b$; Apply: $y_2 = 0$, $y_3 = 3$, $y_4 = 2$; Read R_2/.

To generate a conformity test for the node $m = y_3$, the following paths are activated $l(m) = (y_4, y_3)$, $l(m,1) = (y_3, y_1, R_1+R_2)$, $l(m,2) = (y_3, IN)$, $l(m,3) = (y_3, R_1)$, $l(m,4) = (y_3, y_2, R_1*R_2)$ that produces a test control vector y_1, y_2, y_3, y_4 = 0,0,D,2. The test data vector $DATA = (R^*_1, R^*_2, IN^*)$ is found by solving the inequality $(R_1+R_2) \neq IN \neq R_1 \neq (R_1*R_2)$. The conformity test consists in cyclically run sequence: FOR all $D \in \{0,1,2,3\}$: /Load: $R_1 = R^*_1$; Load: $R_2 = R^*_2$; Apply: $y_1 = 0$, $y_2 = 0$, $y_3 = D$, $y_4 = 2$; $IN = IN^*$; Read R_2/.

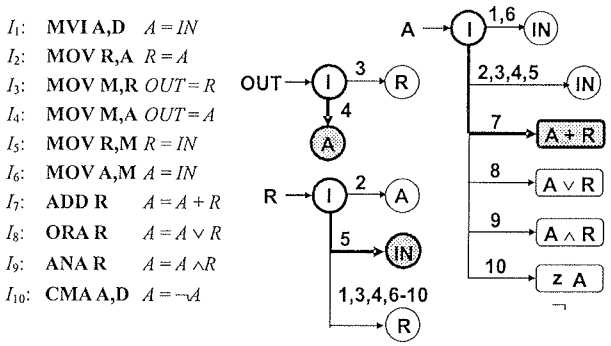

Figure 6: Decision Diagrams for a hypothetical microprocessor

An example of a hypothetical microprocessor is presented in Figure 5 given by a instruction set and three DDs G_A, G_R, G_{OUT}, for representing the behaviour, correspondingly, of accumulator A, register R and output logic. Since the model consists of several DDs representing a network of modules, the following tasks are solved for test generation: fault manifestation, fault propagation and line justification.

To generate a scanning test for the node $A+R$ in G_A, a path $l(A+R) = (I, A+R)$ in G_A is activated, which produces a test pattern $I = I_7$. The test data vector *DATA* = $(A_{1,1},R_{2,1}; A_{1,2},R_{2,2}; ... A_{1,m},R_{2,m})$ for testing the adder are generated by a low level ATPG. These operations correspond to the fault manifestation procedure, i.e. for solving a set of conditions $W^d = 1$ to map the low-level defects of the adder to the behavior level errors in the register A. For propagating the faults from A to OUT, a scanning test for the node A in G_{OUT} is generated. As the result, the path $l(A) = (I, A)$ in G_{OUT} is activated, which produces a test pattern $I = I_4$. For justification of the data variables A and R, the paths $l(IN)$ =

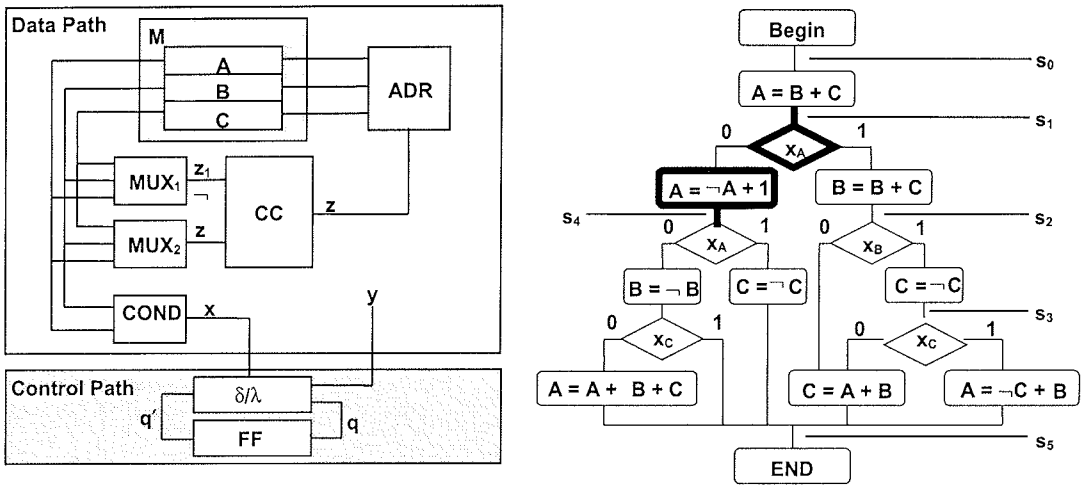

Figure 7: A digital system with a behavioral description

(*I*, *IN*), correspondingly, in G_A and G_R are to be activated which produce symbolic test vectors ($I = I_1$, $IN = a$) and ($I = I_5$, $IN = r$). The scanning test consists in cyclically run sequence: FOR all $(a,r) \in DATA$: $/I_5$: **MOV R,M** (Load $R = r$); I_1: **MVI A,D** (Load $A = a$); I_7: **ADD R** ($A = a + b$); I_4: **MOV M,A** (Read A)/.

Consider a digital system with a behavioral description in Figure 6. The system consists of control and data parts. The FSM of the control part of the system 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 apostrophe refers to the value from the previous clock cycle.

The data path consists of the memory block M with three registers A, B, C together with the addressing block ADR, 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 $x = G_x(A, C)$.

By superpositioning the DDs /21/ we can represent the system by only four DDs G_q, G_A, G_B, and G_C in Figure 7a.

Consider now the possibility of joining a set of DDs into a single DD. In Figure 7b the DDs G_A, G_B, G_C and G_q are joined into a single Vector Decision Diagram (VDD) $M = A.B.C.q = G_M(q', A, B, C, i)$ which produces a new concise model of the system. For calculating the values to different components of the vector variable M, we introduce a new type of node in VDD called *addressing node* labeled by an addressing variable i. The VDDs 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.

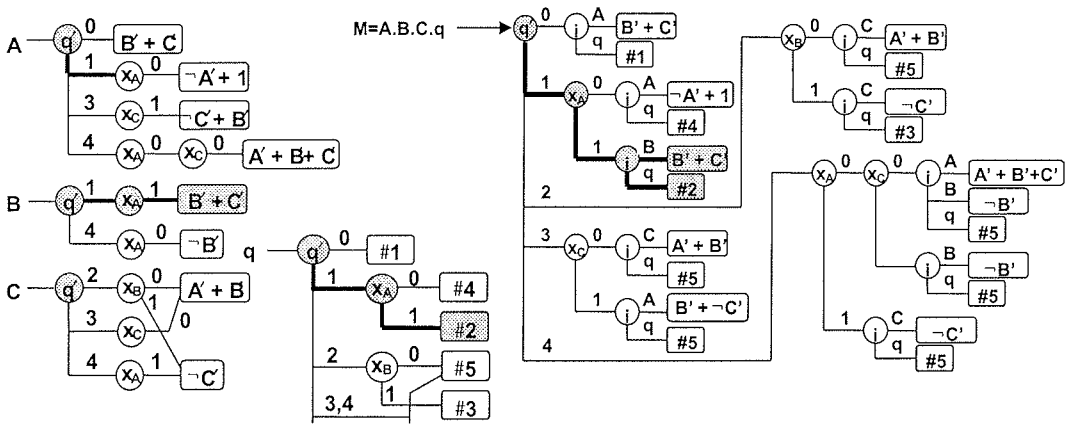

Figure 8. Representing the digital system in Figure 7 by high-level Decision Diagrams

Using VDDs allows significally to increase the speed of simulation. For example, in G_M in Figure 7b 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 in Figure 7a the nodes q' and x_A should be traversed for all the graphs: for calculating separately A, B, C, and q.

5. Experimental results

Table 3 presents the results of investigating the defect-oriented hierarchical test generation based on using of physical defect tables for library cells and logic macro level test generation. Experiments were carried out with a new defect-oriented Automated Test Pattern Generator (ATPG) DOT /22/ for detecting the AND-short defects in the 0.8μm CMOS technology. We used circuits resynthesized from the Verilog versions of the ISCAS85 suite as benchmarks for tests. The circuits were synthesised by SYNOPSYS Design Compiler. Column 2 in Table 3 shows the total number of defects in the defect tables summed over all the gates belonging to the netlist. Column 3 reflects the number of gate level redundant defects. These are defects that cannot be covered by any gate input (GI) vector of the gate. In column 4 circuit level redundant defects are counted. These are defects that cannot be tested as the circuit structure does not allow to generate a test for any of the GI vectors covering the defect. This redundancy is proved by the DOT tool. Column 8 shows the percentage of defects covered by DOT, while column 5 shows the ability of logic level SAF-oriented ATPG to cover the physical defects. The next coverage measure shows the SAF-oriented test efficiency. In this value, both, gate level redundancy of defects (column 6) and circuit level redundancy of defects (column 7) are taken into account.

The experiments prove that relying on 100 % SAF test coverage would not necessarily guarantee a good coverage of physical defects. In many situations the achieved coverage remained well below what can be achievable with the defect-oriented tool. For example, for circuit c2670 the defect coverage 98,29% obtained by SAF tests was more than 1.7 % lower than the result of the proposed tool. An interesting remark is, that up to nearly 25% of the defects were proved redundant by the DOT and can therefore not be detected by any voltage test. 75% defect coverage for c880 by 100% SAF-test gives not much confidence for this test. Only using DOT allows to prove that most of the undetected defects are redundant, and that the real test efficiency of this SAF-test is actually 99,66% giving finally a good confidence to the test.

Table 4: Comparison of ATPGs

Circuit	Faults	HITEC [1]		GATEST [3]		DECIDER [9,23]	
		F.C. %	Time s	F.C. %	Time s	F.C., %	Time, s
gcd	454	81.1	170	**91.0**	75	89.9	**14**
sosq	1938	77.3	728	79.9	739	**80.0**	**79**
mult	2036	65.9	1243	69.2	822	**74.1**	**50**
ellipf	5388	87.9	2090	94.7	6229	**95.0**	**1198**
risc	6434	52.8	49020	96.0	2459	**96,5**	**151**
diffeq	10008	96.2	13320	96.4	3000	**96.5**	**296**
average F.C.:		76.9		87.9		**88.6**	

The experiments of the hierarchical DD-based ATPG DECIDER /9,23/ developed at the Tallinn Technical University were run on a 366 MHz SUN UltraSPARC 60 server with 512 MB RAM under SOLARIS 2.8 operating system. At present, DECIDER contains gate-level EDIF interface which is capable of reading designs of CAD systems CADENCE, MENTOR GRAPHICS, VIEWLOGIC, SYNOPSYS, etc. In Table 4, comparison of test generation results of three ATPG tools are presented on six hierarchical benchmarks. The tools used for comparison include HITEC /1/, which is a logic-level deterministic ATPG and GATEST /3/ as a genetic-algorithm based tool.

Actual stuck-at-fault coverages of the test sequences generated by all the tools were measured by the same fault simulation software. The experimental results show the high speed of the ATPG DECIDER which is explained by the DD-based hierarchical approach used in test generation.

Table 3: Experiments of defect oriented test generation on logic level

Circuit	Number of defects			Defect coverage			
	All defects	Redundant defects		100% stuck-at fault ATPG			DOT
		Gates	System				
1	2	3	4	5	6	7	8
c432	1519	226	0	78,6	99,05	99,05	100,00
c880	3380	499	5	**75,0**	99,50	99,66	100,00
c2670	6090	703	61	79,1	98,29	**98,29**	100,00
c3540	7660	985	74	80,1	98,52	99,76	99,97
c5315	14794	1546	260	82,4	97,73	99,93	100,00
c6288	24433	4005	41	77,0	99,81	100,00	100,00

6. 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 physical 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, structurally synthesized binary decision diagrams, high-level and vector decision diagrams have been discussed. From simulation point of view, they provide a compact and efficient representations of digital systems. 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 IST-2000-30193 REASON and IST-2001-37592 eVIKINGs II, as well as by the Estonian Science Foundation grants 5649 and 5910. A special thanks to my colleagues Jaan Raik and Artur Jutman from Tallinn University of Technology and Joachim Sudbrock from Darmstadt University of Technology (Germany) for developing the prototype tools of test generation and fault simulation by using DDs.

7. References

/1/ T. M. Niermann, J. H. Patel. HITEC: A test generation package for sequential circuits. *Proc. European Conf. Design Automation (EDAC)*, pp.214-218, 1991

/2/ J.F. Santucci et al. Speed up of behavioral ATPG. *30th ACM/ IEEE DAC*, pp. 92-96, 1993.

/3/ E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann. Sequential circuit test generation in a genetic algorithm framework. *Proc. Design Automation Conference*, pp. 698-704, 1994.

/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.Ubar. Test Synthesis with Alternative Graphs. *IEEE Design&Test of Computers*, Spring 1996,pp.48-57.

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

/8/ R.Ubar. Combining Functional and Structural Approaches in Test Generation for Digital Systems. *Microelectronics Reliability*, Vol. 38, No 3, pp.317-329, 1998.

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

/11/ P.Nigh, W.Maly. Layout - Driven Test Generation. *Proc. ICCAD*, 1989, 154-157.

/12/ M.Jacomet and W.Guggenbuhl. Layout-Dependent Fault Analysis and Test Synthesis for CMOS Circuits. *IEEE Trans. on CAD*, 1993, **12**, 888-899.

/13/ R.Ubar, W.Kuzmicz, W.Pleskacz, J.Raik. Defect-Oriented Fault Simulation and Test Generation in Digital Circuits. *2nd Int. Symp. on Quality of Electronic Design*, San Jose, California, March 26-28, 2001, pp.365-371.

/14/ R.Ubar. Test Generation for Digital Circuits with Alternative Graphs. *Proceedings of Tallinn Technical University* No 409, 1976, pp.75-81 (in Russian).

/15/ S.B.Akers. Functional Testing with Binary Decision Diagrams. *J. of Design Automation and Fault-Tolerant Computing*, Vol.2, Oct. 1978, pp.311-331.

/16/ R.Ubar. Vektorielle Alternative Graphen und Fehlerdiagnose für digitale Systeme. *Nachrichtentechnik/Elektronik*, (31) 1981, H.1, pp.25-29.

/17/ R.Ubar. Test Generation for Digital Systems on the Vector Alternative Graph Model. *Proc. of the 13th Annual Int. Symp. on Fault Tolerant Computing, Milano*, Italy, 1983, pp.374-377.

/18/ C.Y. Lee. Representation of Switching Circuits by Binary Decision Programs. *The Bell System Technical Journal*, July 1959, pp.985-999.

/19/ R.Drechsler, B.Becker. Binary Decision Diagrams. *Kluwer Academic Publishers*, 1998, 200 p.

/20/ R.Ubar, A.Moraviec, J.Raik. Cycle-based Simulation with Decision Diagrams. *IEEE Proc. of Design Automation and Test in Europe*. Munich, March 9-12, 1999, pp.454-458.

/21/ R.Ubar, J.Raik, E. Ivask, M.Brik. Hierarchical Fault Simulation in Digital Systems. *Proceedings of Int. Symp. on Signals, Circuits and Systems SCS'2001*, Iasi, Romania, July 10-11, 2001, pp.181-184.

/22/ J.Raik, R.Ubar, J.Sudbrock, W.Kuzmicz, W.Pleskacz. DOT: New Deterministic Defect-Oriented ATPG Tool. *Proc. of 10th IEEE European Test Symposium*, May 22-25, 2005, Tallinn, pp.96-101.

/23/ J.Raik, R.Ubar. Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations. *Journal of Electronic Testing: Theory and Applications. Kluwer Academic Publishers*. Vol. 16, No. 3, pp. 213-226, 2000.

Raimund Ubar
Tallinn University of Technology
Raja 15, 12618 Tallinn, Estonia, raiub@pld.ttu.ee

Prispelo (Arrived): 14. 09. 2005; Sprejeto (Accepted): 28. 10. 2005