# DIAGNOSTIC MODELLING OF DIGITAL SYSTEMS WITH MULTI-LEVEL DECISION DIAGRAMS

R.Ubar, J.Raik, T.Evartson, M.Kruus, H.Lensen,
Tallinn Technical University
Raja 15, 12618 Tallinn
Estonia
{raiub, jaan, teet}@pld.ttu.ee, {kruus, hl}@cc.ttu.ee

**ABSTRACT**
To cope with the complexity of today's digital systems in diagnostic modelling, hierarchical approaches should be used. In this paper, the possibilities of using Decision Diagrams (DD) for diagnostic modelling of digital systems are discussed. DDs can be used for modelling systems at different levels of representation like logic level, register transfer level, instruction set level. The nodes in DDs can be modelled as generic locations of faults. For more precise general specification of faults logic constraints are used. To map the physical defects from transistor level to logic level a new functional fault model is introduced.

**KEY WORDS**
Digital systems, faults and defects, modelling, simulation, test generation, Boolean derivatives, decision diagrams.

## 1. Introduction

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 and high-level modelling. To follow the both trends, hierarchical approaches should be used. One way to manage hierarchy in a uniform way at different levels is to use decision diagrams (DD).

Traditional low-level test 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 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. But, the defect-orientation is increasing even more the complexity. 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 hierarchical approach compared to the 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 languages and models for different levels. Most frequent examples are logic expressions for combinational circuits, state transition diagrams for finite state machines (FSM), abstract execution graphs, system graphs, instruction set architecture (ISA) descriptions, flow-charts, hardware description languages (HDL, VHDL, Verilog etc.), Petri nets for system level description etc. All these models need different manipulation algorithms and fault models which are difficult to merge in hierarchical test methods. Better opportunities for hierarchical diagnostic modelling of digital systems provide Decision Diagrams (DD) [4-9]. Binary DDs (BDD) have found already very broad applications in logic design as well as in logic test [4-5]. Structurally Synthesized BDDs (SSBDD) are able to represent gate-level structural faults directly in the graph [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].

On the other hand, the disadvantage of the traditional hierarchical approaches to test is the traditional 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. To handle physical defects in fault simulation, we still need logic fault models to reduce the complexity of simulation.

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 physical level to logic level. A

generalization of the SAF model called Functional Fault Model (FFM) is presented in Section 3. FFM can be regarded as a uniform interface for mapping faults from a given arbitrary level of abstraction to the next higher level. For hierarchical diagnostic modelling, DDs are used. In Section 4 SSBDDs are presented for logic level test generation and fault simulation. Section 5 explains how the hierarchical approach can be implemented by using higher level DDs. Some experimental data are presented in Section 6 to illustrate the efficiency of the method. Section 7 concludes the paper.

## 2. Modelling Defects and Faults

Consider a Boolean function $y = f(x_1, x_2, \ldots, x_n)$ implemented by an embedded component $C$ in a circuit. Introduce a Boolean variable $d$ for representing a physical defect in the component, which may affect the value $y$ by converting $f$ into another function $y = f^d(x_1, x_2, \ldots, x_n)$. Introduce for the block $C$ a generic parametric function

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

as a function of a defect variable $d$, which describes the behavior of the component simultaneously for both possible fault-free and faulty cases. For the faulty case, the value of $d$ as a parameter is equal to 1, and for the fault-free case $d = 0$. In other words,

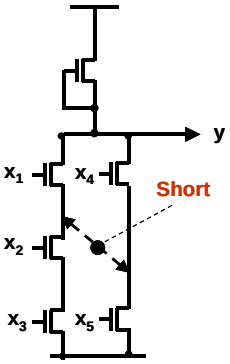



**Fig.1.** Transistor circuit with a short

$$y^* = f^d \quad \text{if } d = 1, \text{ and}$$

$$y^* = f \quad \text{if } d = 0.$$

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 defect $d$ by equations (1) and (2) allows 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. 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.

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 a physical defect in a component can be represented by a logical constraint $W^d = 1$ to be fulfilled for activating the defect. The event of erroneous value on the output $y$ of a 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 a higher level erroneous signal $dy = 1$ iff the condition $W^d = 1$ is fulfilled.

## 3. Hierarchical Approach to Test

The method of defining faults by logic conditions $W^d$ 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, $W^d$ describes how a lower level fault $d$ (a defect either in a component or in the network) should be activated at a higher level to a given node. The conditions $W^d$ can be used both in fault simulation and in test generation.

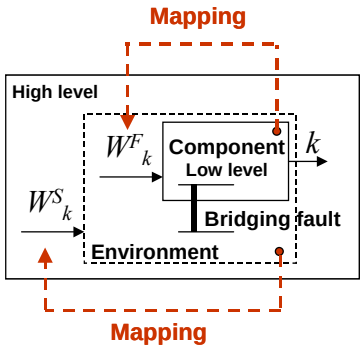



**Fig. 2.** Mapping faults from lower level to higher level

Consider a node $k$ in a circuit (Fig.2) as the output of a module $M_k$, represented by a variable $x_k$.

Associate with $k$ 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 neighbourhood" 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 $d \in R^S_k$.

By using $W^F_k$ and $W^S_k$ we can set up a mapping of faults from a lower level to a higher level for test generation, 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 to be found. In fault simulation (or in fault diagnosis) an erroneous value of $x_k$ (denoted by a Boolean differential $dx_k$=1) can be explained as

$$dx_k \rightarrow d_1 W^{d1} \vee d_2 W^{d2} \vee \ldots \vee d_n W^{dn}$$

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.

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 as explained in Section 2.

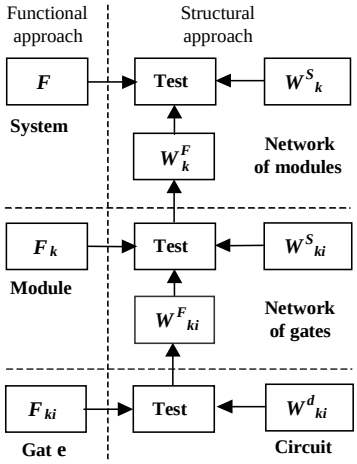



**Fig.3.** Hierarchical approach to test

In Fig.3, a general hierarchical test concept based on parametric fault modeling and the functional fault model for a 3-level system is illustrated.

Consider a task of defect oriented fault simulation in a system which is represented at three levels: RTL, gate and defect levels. Let $Y$ be a RTL variable (an observable point), $y_M$ an output variable of a logic level module and $y_G$ the output of a logic gate with a physical defect $d$, then the condition to detect the defect $d$ on the observable test point $Y$ is

$$W = \partial Y/\partial y_M \wedge \partial y_M/\partial y_G \wedge W^d = 1, \qquad (3)$$

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.

We used the notation $\partial Y/\partial y_M$ to denote the dependency of $Y$ from $y_M$ as an analogue of Boolean derivative for higher level (e.g. RT level) of abstraction for digital systems., dispite of that there is no mathematics available for calculating Boolean derivatives at higher (not logic) levels.

In the following we show how we can calculate Boolean derivatives with BDDs and thereafter how we can generalize this operation for higher level representations based on high-level DDs.

## 4. Modelling Digital Circuits with BDDs

Consider first, the following graph theoretical definitions of the BDD. We use the graph-theoretical definitions instead of traditional *ite* expressions [4,5] because all the procedures defined further for SSBDDs are based on the topological reasoning rather then on graph symbolic manipulations as traditionally in the case of BDDs.

*Definition 1.* A BDD that represents a Boolean function $y=f(X)$, $X = (x_1,x_2, \ldots , x_n)$, is a directed acyclic graph $G_y = (M,\Gamma,X)$ with a set of nodes $M$ and a mapping $\Gamma$ from $M$ to $M$. $M = M^N \cup M^T$ consists of two types of nodes: nonterminal $M^N$ and terminal $M^T$ nodes. A terminal node $m^T \in M^T = \{m^{T,0}, m^{T,1}\}$ is labelled by a constant $e \in \{0,1\}$ and is called *leaf*, while all nonterminal nodes $m \in M^N$ are labelled by variables $x \in X$, and have exactly two successors. Let us denote the associated with node $m$ variable as $x(m)$, then $m^0$ is the successor of $m$ for the value $x(m) = 0$ and $m^1$ is the successor of $m$ for $x(m) = 1$.

*Definition 2.* By the value of $x(m) = e$, $e \in \{0,1\}$, we say the edge between nodes $m \in M$ and $m^e \in M$ is *activated.* Consider a situation where all the variables $x \in X$ are assigned by a Boolean vector $X^t \in \{0,1\}^n$ to some value. The activated by $X^t$ edges form an *activated path* $l(m_0, m^T)$ from the root node $m_0$ to one of the terminal nodes $m^T \in M^T$.

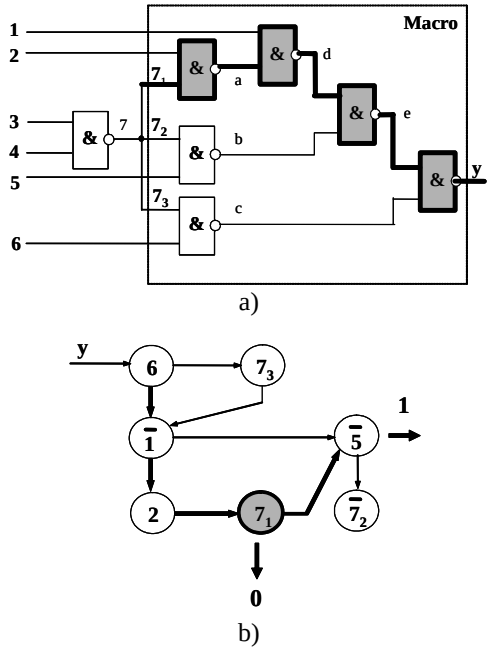



**Fig.4.** Digital circuit and its SSBDD

*Definition 3.* We say that a BDD $G_y = (M,\Gamma,X)$ represents a Boolean function $y=f(X)$, iff for all the possible vectors $X^t \in \{0,1\}^n$ a path $l(m_0, m^T)$ is activated so that $y = f(X^t) = x(m^T)$.

*Definition 4.* Consider a BDD $G_y=(M,\Gamma,X)$ where $X$ is the vector of literals of a function $y = P(X)$ represented in the equivalent parenthesis form [7], the nodes $m \in M^N$ are labelled by $x(m)$ where $x \in X$ and $|M| = |X|$. The BDD is called a *structurally synthesized* BDD (SSBDD) iff there exists one-to-one correspondence between literals $x \in X$ and nodes $m \in M^N$ given by the set of labels $\{ x(m) \mid x \in X, m \in M^N\}$, and iff for all the possible vectors $X^t \in \{0,1\}^n$ a path $l(m_0, m^T)$ is activated, so that $y = f(X^t) = x(m^T)$.

Unlike the traditional BDDs [4-5], SSBDDs [7] support structural representation of gate-level circuits in terms of signal paths. By superposition of DDs [7], we can create SSBDDs with one-to-one correspondence between graph

nodes and signal paths in the circuit. The whole 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 to a higher macro level without loosing accuracy of representing gate-level signal paths.

Fig.4. shows a representation of a tree-like combinational circuit by a SSBDD. For simplicity, values of variables on edges of the SSBDD are omitted (by convention, the right-hand edge corresponds to 1 and the lower-hand edge to 0). Also, terminal nodes with constants 0 and 1 are omitted: leaving the graph to the right corresponds to $y$ = 1, and down, to $y$ = 0. The graph contains 7 nodes, and each of them represents a signal path in a circuit. By bold lines a full activated path is highlighted in the graph corresponding to the input pattern $x_1x_2x_3x_4x_5x_6$ = 110100. The value of the function $y$ = 1 for this pattern is determined by the value of the variable $x_5$ = 1 in the terminal node of the path.

*Procedure 1*. Calculation of Boolean derivatives. To solve a Boolean differential equation $\frac{\partial y}{\partial x(m)} = 1$ for the function $y=f(X)$ with SSBDD $G_y$, where $x \in X$, and $m \in M^N$, the following paths in $G_y$ are to beactivated: 1) $l(m_0, m)$, 2) $l(m^1, m^{T,1})$, 3) $l(m^0, m^{T,0})$.

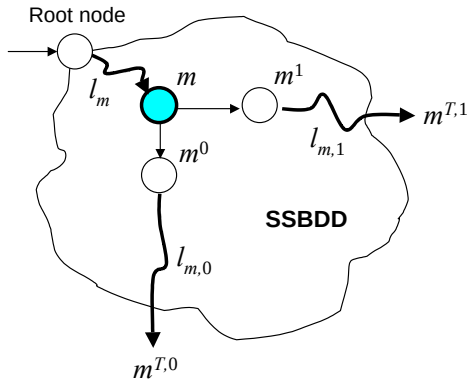



Fig.5. Calculation of Boolean Derivatives with SSBDDs

To solve the Boolean differential equation $\frac{\partial y}{\partial x_{7,1}} = 1$ for the circuit in Fig.4a by using SSBDD means to use the Procedure 1 for the node $m = 7_1$ in the graph in Fig.4b. The following paths should be activated; $(6,\neg 1,2,\ 7_1)$, $(\neg 1,\ m^{T,1})$, and $(\neg 1,\ m^{T,0})$, which produces the pattern: $x_1x_2x_3x_4x_5x_6$ = 11xx00. To test a physical defect of the bridge between the lines 6 and 7, which is activated on the line 7, additional constraints $W=\neg x_6 \wedge x_7=1$ is to be used, which updates the test vector to 111x00.

## 5. Modelling Systems with High Level DDs

Consider now a digital system $S = (Z, F)$ as a network of components where $Z$ is the set of variables (Boolean, Boolean vectors or integers), which represent connections between components, inputs and outputs of the network. Denote by $X \subset Z$ and $Y \subset Z$, correspondingly, the subsets of input and output variables. $V(z)$ denotes the set of possible values for $z \in Z$, which are finite.

Let $F$ be the set of digital functions on $Z$: $z_k = f_k(z_{k,1}, z_{k,2}, \ldots, z_{k,p}) = f_k(Z_k)$ where $z_k \in Z$, $f_k \in F$, and $Z_k \subset Z$. Some of the functions $f_k \in F$, for the state variables $z \in Z_{STATE} \subset Z$, are next state functions.

*Definition 5.* A *decision diagram* (denoted as DD) is a directed acyclic graph $G_k = (M, \Gamma, z)$ where $M$ is a set of nodes, $\Gamma$ is a relation in $M$, and $\Gamma(m) \subset M$ denotes the set of successor nodes of $m \in M$. The nodes $m \in M$ are marked by labels $z(m)$. The labels can be ether variables $z \in Z$, or algebraic expressions of $z \in Z$, or constants.

For non-terminal nodes $m \in M^N$, where $\Gamma(m) \neq \varnothing$, an onto function exists between the values of $z(m)$ and the successors $m^e \in \Gamma(m)$ of $m$. By $m^e$ we denote the successor of $m$ for the value $z(m) = e$. The edge $(m, m^e)$ which connects nodes $m$ and $m^e$ is called *activated* iff there exists an assignment $z(m) = e$. Activated edges, which connect $m_i$ and $m_j$ make up an *activated path* $l(m_i, m_j)$. An activated path $l(m^0, m^T)$ from the initial node $m^0$ to a terminal node $m^T$ is called *full activated path*.

*Definition 6.* A decision diagram $G_k$ represents a function $z_k = f_k(z_{k,1}, z_{k,2}, \ldots, z_{k,p}) = f_k(Z_k)$ iff for each value $v(Z_k) = v(z_{k,1}) \times v(z_{k,2}) \times \ldots \times v(z_{k,p})$, a full path in $G_k$ to a terminal node $m^T \in M^T$ in $G_k$ is activated, so that $z_k = z(m^T)$ is valid.

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 RTL descriptions, we usually partition the system into control and data parts. Nonterminal nodes in DDs correspond to the control path, and they are labelled by state and output variables of the control part serving as addresses or control words. Terminal nodes in DDs correspond to the data path, and they are labelled by the data words or functions of data words, which correspond to buses, registers, or data manipulation blocks. 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. In some simple cases a digital system can be represented as a single DD.

Consider a digital system $z_k=f(Z)$ represented by a single graph $G_k$. We can now generalize Procedure 1 for higher level functions with the goal to create dependencies between high level variables as we used the notation $\partial Y/\partial y_M$ in (3).

*Procedure 2*. Calculating of high-level dependencies of signals (*conformity test*). To make a system level variable $z_k$ depending on an argument $z(m) \in Z$ (denoted as $\frac{\partial z_k}{\partial z(m)} = 1$) for the system function $z_k=f(Z)$ with DD $G_k$.

the following paths are to beactivated: 1) $l(m_0, m)$, 2) for all the values of $e \in v(z(m))$: $l(m^e, m^{T,e})$, and the proper data are to be found by solving the inequality

$$z(m^{T,1}) \neq z(m^{T,2}) \neq \ldots \neq z(m^{T,k}) \text{ where } k = |\, v(z(m))\,|.$$

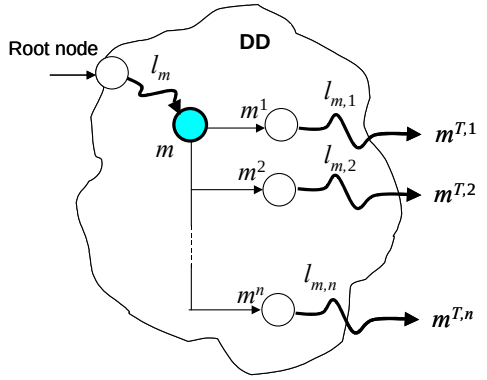
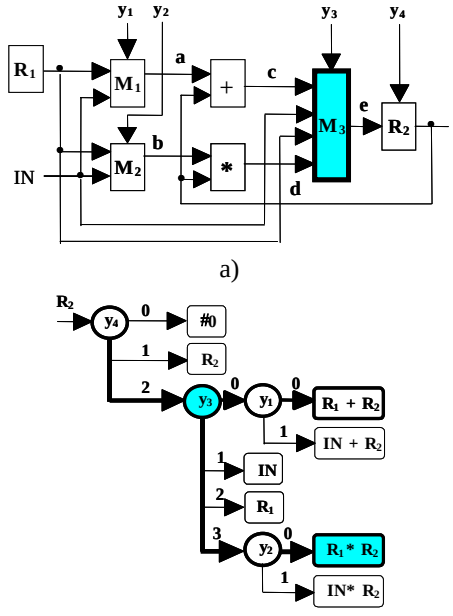



Fig.6. Calculating RT level dependencies with DDs

Solutions find by Procedure 2 are called *conformity tests*. They check if a system is working in a particular working mode defined by the value of a (control) variable $z(m)$ properly.




a)

b)

Fig.7. Representing a data path by a high-level DD

*Procedure 3*. Scanning test. To generate a *scanning test* for a node $m \in M^T$ in $G_y$, the following path is to beactivated: $l(m_0, m)$, and the test patterns for testing the function $z(m)$ should be generated (e.g. on the lower level representation of $z(m)$, according to the hierarchical approach described).

It is easy to notice that the scanning test is a particular case of the conformity test, and results in a similar way as the conformity test from generalization of Procedure 1.

In Fig.7 a RTL data-path and its high-level DD is presented. The variables $R_1$, $R_2$ and $R_3$ represent registers, $IN$ represents the input bus, the integer variables $y_1, y_2, y_3, y_4$ represent the control signals, $M_1, M_2, M_3$ are multiplexers, and the functions $R_1+R_2$ and $R_1*R_2$ represent the adder and multiplier, correspondingly. Each node in DD represents a subcircuit of the system (e.g. the nodes $y_1$, $y_2$, $y_3$, $y_4$ represent multiplexers and decoders,). The whole DD describes the behaviour of the input logic of the register $R_2$. To test a node means to test the corresponding subcircuit.

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 Fig.7 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 activated path). Instead of simulating by a traditional approach all the components in the circuit, in 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.

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

To generate a scanning test for the node $R_1*R_2$ of the DD in Fig.6, a path $l(m_0, m) = (y_4, y_3, y_2, R_1*R_2)$ is to be activated, and the data $DATA = (R_{1,1},R_{2,1}; R_{1,2},R_{2,2}; \ldots R_{1,m},R_{2,m})$ for testing the multiplier are to be generated at low level by an arbitrary 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 $y_3$, the following paths are to be activated $l(m_0, m) = (y_4, y_3)$, $l(m, m^1) = (y_3, y_1, R_1+R_2)$, $l(m, m^2) = (y_3, IN)$, $l(m, m^3) = (y_3, R_1)$, $l(m, m^4) = (y_3, y_2, R_1*R_2)$ that produces a test vector $y_1, y_2, y_3, y_4$ = 0,0,$D$,2. The 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 $R_2$].

## 6. Experimental results

We have carried out two types of experiments: to show the possibility of increasing the accuracy of diagnostic modeling digital circuits by introducing the defect-based functional fault model, and to show the the possibility of increasing the speed of diagnostic modelling by using multi-level DD-based approach.

Table 1: Experiments of defect oriented test generation

| Circuit | Number of defects | | | Defect coverage | | | |
|---|---|---|---|---|---|---|---|
| | All | Redundant defects | | 100% stuck-at fault ATPG | | | New tool |
| | | Gates | Syst | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| c432 | 1519 | 226 | 0 | 78.6 | 99.0 | 99.0 | 100 |
| c880 | 3380 | 499 | 5 | **75.0** | 99.5 | 99.6 | 100 |
| c2670 | 6090 | 703 | 61 | 79.1 | 98.3 | **98.3** | 100 |
| c3540 | 7660 | 985 | 74 | 80.1 | 98.5 | 99.7 | 99.9 |
| c5315 | 14794 | 1546 | 260 | 82.4 | 97.7 | 99.9 | 100 |
| c6288 | 24433 | 4005 | 41 | 77.0 | 99.8 | 100 | 100 |

Table 1 presents the results of investigating the defect-oriented test generation. Experiments were carried out with a new defect-oriented Automated Test Pattern Generator (ATPG) [13]. We used the ISCAS85 suite as benchmarks. Column 2 shows the total number of defects in the fault tables summed over all the gates belonging to the netlist. Column 3 reflects the number of gate level redundant defects. In column 4 circuit level redundant defects are counted. Column 8 shows the number of defects covered by the new ATPG, while column 5 shows the ability of logic level SAF-oriented ATPG to cover physical defects. The next coverage measure shows the 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. For example, for circuit c2670 the defect coverage obtained by SAF tests was more than 1.6 % lower than the result of the proposed tool. An interesting remark is, that up to 25% of the defects were proved redundant by the new tool and can therefore not be detected by any voltage test. 75% of defect coverage for c880 gives not much confidence for this test. Only using the new ATPG allows to prove that most of the undetected defects are redundant, and that the real test efficiency is actually 99,66 giving finally a good confidence to the test.

Table 2: Comparison of ATPGs

| Circuit | Faults | HITEC [1] | | Gatest [3] | | DD-approach | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 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 |
| Diffec | 10008 | 96.2 | 13320 | 96.4 | 3000 | 96.5 | 296 |
| Average FC | | 76.9 | | 87.9 | | 88.6 | |

The experiments of the DD-based ATPG for digital systems were run on a 366 MHz SUN UltraSPARC 60 server with 512 MB RAM under SOLARIS 2.8 operating system. The system contains gate-level EDIF interface which is capable of reading designs of CAD systems CADENCE, MENTOR GRAPHICS, VIEWLOGIC, SYNOPSYS, etc. In Table 2, comparison of test generation results of three ATPG tools are presented on 6 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. The experimental results show the high speed of the new ATPG tool which is explained by the efficient algorithms based on DDs and by the hierarchical approach used in test generation

## 7. Conclusion

A method for modelling defects by a functional fault model was developed as a general basis for hierarchical approach to digital test. For hierarchical diagnostic modelling of digital systyems, multi-level DDs were proposed as an efficient model for representing digital systems in a uniform way at different representation levels.

*Acknowledgements:* This work has been supported by the Estonian Science Foundation grants 5649 and 5910.

## 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. of DAC*, 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 DATE*. 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 and 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] J.Raik, R.Ubar, J.Sudbrock, W.Kuzmicz, W.Pleskacz. DOT: New Deterministic Defect-Oriented ATPG Tool.. Proc. of the 10th IEEE European Test Symposium. Tallinn, May 22-25, 2005.