Hierarchical Calculation of Malicious Faults for Evaluating the Fault-Tolerance

Raimund Ubar, Sergei Devadze, Maksim Jenihhin, Jaan Raik, Gert Jervan, Peeter Ellervee
Tallinn Technical University, Raja 15, 12618 Tallinn, Estonia
{raiub, serega, maksim, jaan, gerje, lrv}@pld.ttu.ee

Abstract
In this paper, a new hierarchical multi-level technique for malicious fault list generation for evaluating the fault tolerance is presented. For the description of the system three levels are exploited: behavioral, functional signal path and structural gate-network levels, whereas at each level the model of decision diagrams and uniform fault analysis procedures are used. Malicious faults are found by top-down technique, keeping the complexity of candidate fault sets at each level as low as possible.

1. Introduction
Dependable systems are designed with fault tolerance features to first, detect errors and then to mask or recover from the effects of those errors. Thus, testing of these features is extremely important in understanding how dependable the systems are with the incorporated fault tolerance mechanisms and in gaining insight into the success of error detection and recovery. Fault injection is a means to effectively test and stress the error handling and fault tolerance mechanisms, so that the system behavior can be studied prior to their actual deployment.

Fault injection techniques can be classified in three main categories [1]: physical (or hardware implemented fault injection), software implemented and simulation-based. Simulation-based fault injection is a useful experimental way to evaluate the dependability of a system during the design phase.

Many simulation based fault injection techniques have been proposed for system dependability evaluation [2-6]. They are targeted for execution-based models of working systems. Simulating faults in a system model (e.g. based on hardware description language VHDL, Verilog, System C etc) assures high flexibility.

However, there exists a problem of selecting faults to be injected. Erroneous responses in a system, in many cases, do not necessarily lead to a failure at the application level, even when the discrepancy with the nominal behavior has a long duration. An accurate but high-level fault analysis in the complete system is therefore required to discriminate real failure conditions from non-critical errors. Such an analysis is very difficult to carry out on the execution-based models using languages like VHDL, Verilog, System C.

In this paper a method is proposed based on multi-level modelling of systems with Decision Diagrams (DD) to select faults for injection targeting dependability evaluation. The system is simulated at higher RT-level with high-level DDs, and at lower logic level with structurally synthesized Binary Decision Diagrams (SSBDD). In such a way the efficiency of high-level calculation is combined with the accuracy of low-level fault analysis.

The paper is organized as follows: Section 2 gives an overview about the malicious faults and fault injection. Section 3 introduces the model used for fault analysis. Section 4 describes the method of malicious fault generation, Section 5 presents experimental data, and in Section 6 some concluding remarks are given.

2. Malicious Faults and Fault Injection
Traditionally in simulation-based fault injection techniques, the fault location, fault type and fault insertion time are typically selected at random. The drawback of randomly selected faults is the large probability that the injected fault will remain latent; that is, produce a no response fault injection experiment. This will reduce the quality of dependability evaluation procedure. Thus, there is a need to locate faults which do not belong to the no response category.

The goal of the fault injection experiment is to exercise the system’s fault processing capabilities. Faults which fail a system in the absence of system fault detection capabilities are defined to be malicious [8,9]. Malicious faults systematically exercise the fault processing attributes of the system. A malicious fault if undetected will fail the system under test. Likewise, a malicious fault is guaranteed to produce an error which will produce a failure if it is not properly processed by the system. Thus, using malicious faults to estimate the dependability of the system eliminates the possibility of the no response fault injection experiment.

Malicious fault list can be generated by creating fault trees using fault dependency analysis by reverse implication and providing fault collapsing. To cope with
the complexity of fault analysis when carried out at plain low (logic) levels, we propose a hierarchical approach where the fault tree creation and fault collapsing will be carried out first at a higher level, thereafter to refine the fault injection points at a lower level.

3. High-Level Decision Diagrams

DDs allow to investigate and solve the problems of fault dependency analysis by a uniform way at different abstraction levels of digital systems. The well known Binary Decision Diagrams (BDD) [10] can be considered as a particular case of this model for using at logic level.

Consider 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 1.** 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 either variables \( z \in Z \), or algebraic expressions of \( z \in Z \), or constants.

For non-terminal nodes \( m \), where \( \Gamma(m) \neq \emptyset \), an onto function exists between the values of \( z(m) \) and the successors \( m^* \in \Gamma(m) \) of \( m \). By \( m^* \) we denote the successor of \( m \) for the value \( z(m) = e \). The edge \((m, m^*)\) which connects nodes \( m \) and \( m^* \) is called activated if 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 \( l(m^*, m^{'}) \) from the initial node \( m^* \) to a terminal node \( m^{'}, \) is called full activated path.

**Definition 2.** 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 \( m^{'}, \) is activated, so that \( z_k = z(m^{'}) \) is valid.

Each function \( f_k(Z_k) \in F \) of the system \( S = (Z, F) \) is represented by a decision diagram \( G_k: z_k = G_k(Z_k), z_k eZ, Z_k \subset Z \) [11-13].

For lower (logic) level fault analysis we will use a special class of BDDs – structurally synthesized BDDs (SSBDD) [11] to represent the topology of digital circuits in terms of signal paths at the fanout free region (or macro) level. Macro level fault analysis can be carried out more efficiently than traditional gate level analysis [13].

In high-level descriptions, we usually partition the system into high level components or subsystems where the descriptions of subsystems in general case can be modeled by control and data paths. These paths can be represented by a set of DDs where the nonterminal nodes in DDs correspond to the control variables (instruction codes, addresses, control words, logic or timing conditions etc), and the terminal nodes correspond to data variables, constants, functional expressions or more complex behavioral or algorithmic descriptions of other subsystems. The terminal nodes will be represented again by a set of DDs to allow entering a lower hierarchical level for disclosing the label of the terminal node [12,13]. Consider in the following without loss of generality finite state machines (FSM) as a subclass of digital systems at three levels – behavioral state transition diagram (STD), functional signal path and gate-network levels by using

![State transition diagram and HLDD for a finite state machine](image-url)
only the DD model.

In Fig.1 a state transition diagram of a FSM and its equivalent as a high-level DD (HLDD) is represented. The HLDD represents the complex behavior function of the FSM \( q.y = F(q', r, x_1, x_2) \) where \( q.y \) is the concatenation of the integer state variable \( q \in \{1,2,3,4,5,6\} \) and the binary output variable \( y \). Input of the FSM is structured and represented by three Boolean variables \( r, x_1 \) and \( x_2 \). By \( q' \) we denote the previous state variable. In the case of FSM, the terminal nodes of the HLDD are labelled by complex (concatenated) constants which represent the new state \( q \) of the FSM and the value of the output variable \( y \) at the new state. Differently from the usual form of STD, to be able formally model the faulty behavior of the FSM, we have to specify in HLDDs also the behavior of FSM at illegal states, denoted by \( q = * \). In Fig.1, for illegal states, it has been assumed \( y = 0 \). In general case the terminal nodes can be labelled by high-level variables (registers) or data manipulation functions [12,13].

Assume that the state variable \( q \) of the FSM in Fig.1 is represented by a state register consisting of three flip-flops \( (T_1, T_2, T_3) \). An example of a next-state logic for the flip-flop \( D_3 \) of the state register, and its SSBDD are depicted in Fig.2. Each node in the SSBDD with a label \( z(m) \in \{T_1, T_2, T_3, x_1, x_2\} \) is representing a signal path in the circuit from the input \( z(m) \) to the output \( D_3 \) (the mapping is shown by numbers). For simplicity, the values of variables on the edges of SSBDD are omitted (by convention, the right-hand edge corresponds to 1 and the lower-hand edge to 0). Also, the terminal nodes with constants 0 and 1 are omitted (leaving the graph to the right corresponds to \( D_3 = 1 \), and down to \( D_3 = 0 \). The node variables are inverted if the number of invertors in the corresponding path is odd.

SSBDD in Fig.2 is a functional signal path model of the next state logic for the flip-flop \( T_3 \) of the FSM represented in Fig.1.

The lowest representation level is the gate network shown by the circuit in Fig.2. As an example, a single node 8 (bold circle) in the SSBDD represents the whole bold path from one of the fanouts of the input \( T_2 \) along the path 8 up to the output \( D_3 \) of the circuit. The set of stuck-at-1 faults along this path in the circuit is represented in the SSBDD by only a single representative fault \( T_2 = 1 \) (T2 stuck-at-1, i.e. the edge from the DD node 8 is constantly activated to the right direction). In this way, the transition from the gate-level model to the corresponding SSBDD is equivalent to a procedure of fault collapsing: all the 10 faults on the signal path 8 in the circuit are collapsed into the 2 faults of the node 8 in the SSBDD.

The activated paths in the SSBDD (shown by bold arrows) represent the situation when the fault \( T_2 = 1 \) is activated. This is the case of the test pattern 0011- (\( T_1, T_2, T_3, x_1, x_2 \)), where "-" denotes the "don't care".

**4. Malicious Fault Analysis with DDs**

The faults having effect on the operation associated with activated path of DD can be related to the nodes along the path. A fault may cause incorrect leaving of the activated path. From this point of view we can introduce a very general fault model for DDs related to faulty behavior of nodes. In the case of SSBDDs the faults of a binary node are equivalent to the traditional logic level stuck-at-0(1) fault model.

In more general case of DDs the faults of the nonterminal nodes can be interpreted as the addressing fault model of microprocessor cores [13]: output edge is either always activated (stuck) or always broken, or instead of the given edge, another edge or a set of edges is activated. This is a very general fault model that

---

**Fig. 2. Combinational circuit and its SSBDD**
allows to represent most of control errors in the case of microprocessor cores.

It is easy to extend the described fault model of a DD node to model more complex faults like shorts, delays, crosstalks etc. All of these faults can be represented by a checkpoint where the error manifests at certain conditions i.e. by a pair (fault site; fault condition). Since the nodes of DDs can be regarded as checkpoints, the fault model on DDs can be represented as a pair (node; fault condition). In such a way the fault model on DDs can be seen as a very powerful model able to cover a very wide class of faults in digital systems.

The formalism of DDs allows to implement simple algorithms to analyze the cause-effect and fault dependency relationships which would be impossible in case of descriptions written in languages like VHDL, Verilog, System C.

Malicious fault list will be generated for a given set of test sequences. A test activates a process in a system consisting of a sequence of iterations. Each iteration (clock cycle, microinstruction or instruction cycle, transaction etc.) activates paths in DDs of the corresponding DD-model of the system. The faults having effect on the behavior of the system at a given iteration can be related to the nodes of the activated path. For each observable checkpoint $C_i$ of a test sequence, a Fault Tree (FT) with a root $N_i$ will be created. Assuming an erroneous signal is detected at $C_i$, a set of candidate faulty checkpoints $C_k$ explaining the misbehavior in $C_i$ is created by tracing the activated path on the corresponding DD for $C_i$. This set of checkpoints $C_k$ will be included into FT as successors of $N_i$. For each node $N_i$ in FT, a list of malicious faults associated with $C_i$ is determined. For all leaf nodes in FT to be constructed, the same procedure is repeated until the input events for the test sequence are reached. If $C_i$ represents stored data (register variables), we will continue fault dependency analysis in the previous test iteration. Each node in FT characterised with a list of malicious faults and the number of iteration can be used as a point for fault injection. To reduce the number of malicious faults, the FT can be compressed (collapsed) by using fault equivalence and fault dominance relationships. The fault model is not restricted by the approach. In general, arbitrary changes $R + \varepsilon$ of the values of a variable $R$ can be accepted. The set of accepted values of $\varepsilon$ can be given by a fault list.

**Example.** Consider a FSM in Fig.3 as a higher level structure, consisting of three next-state logic blocks $KS_i$, $i = 1,2,3$ and the state register of 3 flip-flops. The behavior of the FSM is described by HLDD. By bold arrows in the DD, an activated path is depicted which corresponds to the test pattern 101- ($q', Res, x_1, x_2$). This pattern can detect, for example, the following error at $q$: instead of the expected value 2 we will have 3. Denote such a fault as $q: 2 \leadsto 3$. The activated path passes nodes 1,3,4, which all are included into the fault tree as the possible causes of the fault $q: 2 \leadsto 3$. A simple analysis on the DD shows that only changes at nodes 3 ($q' = 5$) would cause the misbehavior.

![Fig. 3. Malicious fault analysis in FSM at the functional STD level with HLDD](image-url)
and 4 ($x_1 = 0$) can explain this faulty behavior of the FSM. Hence, the nodes 3 and 4 could serve as malicious fault injection sites. However, we will continue the cause-effect fault analysis on the lower level model with the goal to collapse the higher-level faults at the node 3. The node 4 can be fixed already as the fault injection site, because it represents the input, and the further fault analysis at the lower level with the goal of fault collapsing is not needed. The faults at the node 1 (Res) will be excluded from further analysis because they cannot be the causes of the fault q: 2$\rightarrow$3.

Descending to the bit-level, we can find that the lower level fault in the 3rd bit of the state code is causing the higher level fault q: 2$\rightarrow$3, which means that we have to start the lower level fault analysis in the next-state logic for the flip-flop $D_3$ presented in Fig.4.

Consider now the gate-level implementation of the block $KS_3$ and its SSBDD-model in Fig.4.

The activated bold path on the SSBDD corresponds to the test pattern 00101- ($T_1,T_2,T_3,Res,x_1,x_2$) to be analysed, at which the error in the value of $D_3$ can be detected. The activated path goes through nodes 0,1,7,8,11,12,13, which all will be included into the fault tree. A simple reasoning on the SSBDD shows that only changes at nodes 0 (Res = 1), 8 ($T_2 = 1$) and 13 ($x_1 = 0$) can explain the higher level fault q: 2$\rightarrow$3 (or now the corresponding lower level fault $D_3 = 1$). Using the knowledge from the higher level analysis that the faults at Res cannot explain the fault q: 2$\rightarrow$3, and that the node $x_1$ is already fixed as the fault injection site, only the node 8 is selected as the new additional site for fault injection. The faults $x_1 = 0$ and $T_2 = 1$ are malicious faults, because if the fault tolerant mechanism will not work they will cause the error q: 2$\rightarrow$3. The whole reasoning procedure along the fault trees takes 3 steps on the HLDD, and 7 steps on the SSBDD (in total 11 steps). If we carried out the reasoning gate by gate on the plain gate-level we would need in this example to analyze 17 nodes in the next-state logic for flip-flop $D_3$ in Fig.4. In general case all the three next-state circuits may be the objectives for fault probing.

5. Experimental results

Table 1 presents the fault simulation results for the circuits of ISCAS’85 benchmark family (column 1) to compare different fault simulators: two state-of-the-art gate-level commercial fault simulators C1 and C2 from major CAD vendors (columns 2, 3), and the proposed new SSBDD based macro-level fault simulator (column 4) where the fanout free regions were represented as macro-level SSBDDs. The simulation times were calculated for the sets of random 10000
patterns. Compared to the commercial tools C1 and C2, the average gain in speed is 20.8 and 2.8 times, respectively.

Table 2 presents run-times of the high-level DD models and corresponding VHDL models simulation. The experiment was run on a 366 MHz SUN UltraSPARC 60 workstation with 512 MB RAM under Solaris 2.5.1 operating system. During the experiments, real test stimuli generated by test generator DECIDER [14] were used in order to activate all possible states of the circuit behavior (in contrast to random simulation vectors, which in reality do not allow to simulate all possible behaviors). For optimal performance, Synopsys tools cylab and cysim were run with -perf and -2state options. The DD event-driven cycle-based simulation tool implementation offers the gain in simulation time between 2.5 and more than 14 times in comparison to the cycle-based HDL simulator.

Table 1. SSBDD and gate-level simul. comparison

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Fault simulation time [s]</th>
<th>C1</th>
<th>C2</th>
<th>SSBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td></td>
<td>13.0</td>
<td>3.8</td>
<td>0.77</td>
</tr>
<tr>
<td>c499</td>
<td></td>
<td>3.0</td>
<td>2.8</td>
<td>1.91</td>
</tr>
<tr>
<td>c880</td>
<td></td>
<td>26.0</td>
<td>4.0</td>
<td>0.89</td>
</tr>
<tr>
<td>c1355</td>
<td></td>
<td>44.0</td>
<td>9.0</td>
<td>3.28</td>
</tr>
<tr>
<td>c1908</td>
<td></td>
<td>53.0</td>
<td>15.6</td>
<td>3.36</td>
</tr>
<tr>
<td>c2670</td>
<td></td>
<td>104.0</td>
<td>11.0</td>
<td>2.76</td>
</tr>
<tr>
<td>c3540</td>
<td></td>
<td>191.0</td>
<td>37.4</td>
<td>6.02</td>
</tr>
<tr>
<td>c5315</td>
<td></td>
<td>246.0</td>
<td>28.6</td>
<td>8.56</td>
</tr>
<tr>
<td>c6288</td>
<td></td>
<td>1159.0</td>
<td>139.2</td>
<td>72.24</td>
</tr>
<tr>
<td>c7552</td>
<td></td>
<td>378.0</td>
<td>40.5</td>
<td>10.09</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>221.7</td>
<td>29.2</td>
<td>10.6</td>
</tr>
</tbody>
</table>

Table 2. DD and HDL-based simulation comparison

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Simulation time [s]</th>
<th>DD event-driven cycle-based</th>
<th>HDL cycle-based</th>
<th>Ratio HDL/DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd</td>
<td></td>
<td>0.20</td>
<td>0.51</td>
<td>2.55</td>
</tr>
<tr>
<td>mult8x8</td>
<td></td>
<td>0.32</td>
<td>1.00</td>
<td>3.13</td>
</tr>
<tr>
<td>diffeq</td>
<td></td>
<td>0.25</td>
<td>1.26</td>
<td>5.04</td>
</tr>
<tr>
<td>huff enc</td>
<td></td>
<td>0.34</td>
<td>1.40</td>
<td>4.12</td>
</tr>
<tr>
<td>circ1</td>
<td></td>
<td>0.14</td>
<td>2.05</td>
<td>14.64</td>
</tr>
</tbody>
</table>

6. Conclusions

The novelty of the work lies in using DDs for multi-level malicious fault list generation, which makes possible hierarchical approach to cope with the complexity problems. The experimental results demonstrate the advantages of the proposed approach where a considerable speed-up was shown, first in comparison between the macro and gate-level simulation, and secondly, between the RT level and logic level simulation.

Acknowledgements

The work has been supported by Estonian Science Foundation grants 5910, 6717, 6829 7068, EC 6th FW IST project VERTIGO, Estonian Inf. Technology Foundation (EITSA) and Enterprise Estonia.

References