Formal Robustness Checking

Görschwin Fey Rolf Drechsler

University of Bremen, 28359 Bremen, Germany
{fey,drechsle}@informatik.uni-bremen.de

Abstract. Correct input/output behavior of circuits in presence of internal malfunctions becomes more and more important. But reliable and efficient methods to measure this *robustness* are not available yet.
In this paper a formal measure for the robustness of a circuit is introduced. Then, an algorithm to determine the robustness is presented. This is done by reducing the problem either to sequential equivalence checking or to a sequence of property checking instances. The technique also identifies those parts of the circuit that are not robust from a functional point of view and therefore have to be hardened during layout.

1 Introduction

The number of safety critical applications that rely on integrated circuits is growing, e.g. "stear-by-wire" in cars or important control functions in planes. The functional correctness of these circuits is certified by massively applying simulation-based as well as formal verification methods.

At the same time the number of components integrated in a single circuit grows rapidly according to Moore's Law. Meanwhile the physical area occupied by a single component shrinks continuously. As a result a circuit that is functionally correct becomes sensitive to faults that occur after production during in-field application. Among such faults are transient malfunctions due to environmental radiation that cause *Single Event Upsets* (SEU) or static faults caused e.g. by electro-migration due to aging of the material.

Architectural measures are already applied during circuit design to ensure that malfunctions of individual components do not impact the functional correctness. Instead the malfunction is signaled while the input/output behavior is consistent with the original specification. A simple technique to achieve such robustness is the redundancy of functional components. Fault tolerant codes are more sophisticated.

A symbolic approach to analyze the reliability of circuits has recently been introduced in [1]. Outcome of the analysis is a probability for faults in the output response of the circuit. Hints to identify internal structures of the circuit that are not robust are not provided.

Simulation-based validation techniques are commonly used to ensure that the circuit fulfills the specification even in presence of malfunctions. These malfunctions are injected into the internal structures of the circuit. Then, simulation shows whether the malfunction propagates faults to the outputs. To improve

the coverage of the state space, emulation techniques can be applied [2]. But these techniques are incomplete in the sense that not all states of the system can be covered. States that cause faulty behavior when a malfunction occurs may remain uncovered.

On the contrary, the application of formal methods proves that *any malfunction* in *any state* of the system *under any input sequence* (1) is detected and (2) does not cause erroneous input/output behavior. First approaches for such techniques were proposed in [3, 4]. Both methods apply tools for formal verification as a "black box". To prove the robustness of a circuit with respect to a given fault model, each individual fault has to be injected by applying a *mutant* to the circuit description. The resulting faulty circuit is then formally verified against the correct circuit. Therefore in both cases an explicit enumeration of all possible faults is necessary, which is not feasible to capture multiple faults occurring at the same time. Moreover, the set of faults covered by the methods is limited by the mutants that are applied. The method proposed in [3] only returns a "yes" or "no" to the question whether the circuit is robust with respect to a particular fault. Additionally, [4] determines the percentage of "robust states" of the system. Unfortunately, none of the answers is very helpful when trying to identify the parts of the circuit where the robustness has to be improved by architectural changes or by hardening the physical circuit structures [5].

Here, a formal approach is presented to implicitly consider all faults with respect to three formally defined fault models. The proposed algorithm determines those locations in the circuit where the fault tolerance has to be improved. For each location that is not robust, a particular fault and a simulation trace that excites the faulty output response can be calculated. Moreover, the robustness of a circuit with respect to a given fault model is formally defined. When 100% robustness are achieved, no fault of the given fault model has an impact on the input/output behavior of the circuit. The calculation of the robustness measure is reduced to sequential equivalence checking. On the basis of formal methods, the process to implicitly consider all faults is explained. A solver for *Boolean Satisfiability* (SAT) is applied as the proof engine. The basic technique has some similarities to SAT-based diagnosis as introduced by [6]. Finally, techniques to improve the performance of the algorithm are presented and discussed. The practical applicability is shown by empirical studies.

This paper is structured as follows: The preliminaries are briefly discussed in the following section. Next, the notion of robustness is introduced together with the appropriate fault models in Section 3. The approach to implicitly consider all faults according to a given fault model is presented in Section 4. A reduction to a sequence of model checking instances and other techniques to improve the performance are proposed in Section 5. First experimental results are reported in Section 6. Finally, the work is summarized in the last section.

2 Preliminaries

In the following circuits are considered. A *circuit* $\mathcal{C}$ consists of a set of components. Among these are primary inputs, primary outputs, state elements and internal combinational components $g \in \mathcal{C}$. A Boolean function is associated with each internal component. A single gate, a module or a *Register Transfer* (RT) level expression may correspond to a component. The structure of the circuit is defined by a graph. In particular, this graph uniquely provides predecessors and successors of a component.

The *size* of the circuit is given by the number of components, i.e. by $|\mathcal{C}|$. A *part* of a circuit is a subset $\mathcal{S} \subseteq \mathcal{C}$ of the components, the size of which is given by $|\mathcal{S}|$.

The input/output behavior of the circuit emerges from the composition of components and their functionality. Starting from a defined initial state, that is reached by a reset sequence, a particular input sequence leads to a unique output sequence [7].

For the manipulation of Boolean functions there exist different techniques. Among these are *Binary Decision Diagrams* (BDDs) [8] or SAT provers [9, 10]. In this work SAT provers are applied. The transformation of a circuit into a SAT instance requires runtime and memory resources linear in the size of the circuit [11, 12]. The decision whether a SAT instance is satisfiable is NP-complete [13]. Nonetheless, modern SAT solvers solve very efficiently problem instances derived e.g. during formal verification or test pattern generation [14, 15, 10].

3 Measuring Robustness

Fault models are introduced in this section and motivated by faults of practical relevance. Then, a formal measure of robustness is defined with respect to the fault models.

3.1 Fault Models

Several types of faults occur that change the functionality of circuits during in-field application. These faults can be grouped in transient faults, e.g. so called SEUs caused by radiation, and static faults, e.g. due to electro-migration processes. To differentiate the robustness of a circuit with respect to these realistic types of faults, appropriate fault models are introduced in the following.

Definition 1. *A circuit $\mathcal{C}$ and a part $\mathcal{S} \subseteq \mathcal{C}$ of this circuit are given.*

1. *Injecting a fault according to the* non-deterministic fault model $\mathcal{F}_N$*, means to replace the outputs of a component $g \in \mathcal{S}$ by new primary inputs.*
2. *Injecting a fault according to the* combinationally deterministic fault model $\mathcal{F}_C$*, means to replace a component $g \in \mathcal{S}$ by a new combinational subcircuit that has the same successors as g.*

3. *Injecting a fault according to the* locally deterministic fault model $\mathcal{F}_L$*, means to replace a component* $g \in \mathcal{S}$ *by a new combinational circuit that has the same predecessors and successors as* g*.*

Remark 1. Note that the sequence of fault models $\mathcal{F}_N$, $\mathcal{F}_C$ and $\mathcal{F}_L$ imposes an increasing number of constraints onto the functional modification of the circuit. For example, each faulty output response that can be achieved by injecting a fault according to $\mathcal{F}_C$, can also be created by injecting a fault according to $\mathcal{F}_N$ – but not vice versa.

The fault models correspond to different realistic fault types. For example, SEUs can be modeled as non-deterministic behavior defined by $\mathcal{F}_N$.

In the following the set $\mathbb{C}_{\mathcal{C},\mathcal{S},\mathcal{F},N}$ denotes the set of all circuits that can be derived from circuit $\mathcal{C}$ by injecting N faults according to fault model $\mathcal{F}$ into the part $\mathcal{S} \subseteq \mathcal{C}$.

3.2 Definition

A circuit is called robust if no fault changes the input/output behavior. Nonetheless, for example a SEU that occurs at a primary output of a circuit may inevitably modify the output response of the circuit. To avoid this, individual parts of a circuit can be hardened during fabrication, e.g. by using larger structures to realize the components. But this kind of robustness cannot be captured on a Boolean model of the circuit without layout or mapping information. Therefore a more sophisticated definition of robustness that can be applied to parts of the circuit is necessary.

Moreover, in some cases robustness with respect to single faults may not be sufficient, because even a local phenomenon may cause a malfunction of multiple components. Therefore the notion of robustness is defined with respect to multiple faults as well.

Both aspects – the consideration of parts of a circuit and multiple faults – are covered by the following definitions.

Definition 2. *A circuit* $\mathcal{C}$*, a fault model* $\mathcal{F}$ *and an integer* $N \geq 1$ *are given.*

A part $\mathcal{S} \subseteq \mathcal{C}$ *of* $\mathcal{C}$ *is called* $(\mathcal{F}, N)$-robust *if no injection of* N *faults into* $\mathcal{S}$ *according to* $\mathcal{F}$ *changes the input/output behavior of* $\mathcal{C}$*.*

On this basis a formal measure for the robustness of a circuit $\mathcal{C}$ for N-fold faults with respect to a fault model $\mathcal{F}$ can be given. Using the largest $(\mathcal{F}, N)$-robust part S of the circuit is in general not sufficient in presence of multiple faults because some other part T that is not $(\mathcal{F}, N)$-robust may share components with S. Therefore the largest part $\mathcal{S}$ of $\mathcal{C}$ is determined that does not have a component which occurs in an N-fold fault that changes the input/output behavior of $\mathcal{C}$. This is formalized by the following definition.

Definition 3. *A circuit $\mathcal{C}$, a fault model $\mathcal{F}$ and an integer $N \geq 1$ are given. The* $(\mathcal{F}, N)$-robustness *of $\mathcal{C}$ is given by $R_{\mathcal{F},N} = \frac{|\mathcal{S}|}{|\mathcal{C}|}$, where $\mathcal{S}$ is a maximal subset of $\mathcal{C}$ such that forall $\mathcal{T} \subseteq \mathcal{C}$ if*

$$\mathcal{S} \cap \mathcal{T} \neq \emptyset \text{ and } |\mathcal{T}| \leq N$$

then

$$\mathcal{T} \text{ is } (\mathcal{F}, |\mathcal{T}|)\text{-robust}$$

Remark 2. The robustness of a circuit with respect to a given formal property can be defined analogously. Accordingly, the algorithm that is introduced in the next section can by applied to calculate the robustness with respect to a property.

4 Calculating Robustness

4.1 Reduction to Sequential Equivalence

The calculation of the robustness of a circuit can directly be mapped to sequential equivalence checking.

Theorem 1. *A circuit $\mathcal{C}$ and a set of faulty circuits $\mathbb{C}_{\mathcal{C},\mathcal{S},\mathcal{F},N}$ are given. A part $\mathcal{S}$ is $(\mathcal{F}, N)$-robust if and only if each circuit $\mathcal{C}' \in \mathbb{C}$ is sequentially equivalent to $\mathcal{C}$.*

Despite the direct mapping of state elements between faulty circuit and original circuit, a simple reduction to combinational equivalence is not possible in general. A fault may change the state transition function without impact on the input/output behavior. Moreover, the number of derived faulty circuits is very large. Therefore an enumeration of all these circuits would be too time consuming. For this reason an algorithm to consider all faulty circuits in a single instance of Boolean Satisfiability is presented in the following.

4.2 Implicit Enumeration of All Faults

The proposed approach borrows ideas that were originally proposed for diagnosis based on Boolean satisfiability [16, 17]. During diagnosis a modification of the circuit is needed that allows to correct faulty behavior. In the context of robustness checking, a modification that causes incorrect behavior is required.

Initially, the approach is explained at hand of fault model $\mathcal{F}_N$ and then extended to handle the other fault models. The creation of the SAT instance is explained in terms of a circuit that is transformed into conjunctive normal form afterward.

Figure 1 shows the overall flow in pseudo code. The algorithm determines the robustness of a circuit $\mathcal{C}$ with respect to fault model $\mathcal{F}$ and N-fold faults as described in Section 3.2. For this purpose at first all non-robust parts up to size N are determined, collected and then $\mathcal{S}$ is calculated. First a copy $\mathcal{C}'$ of $\mathcal{C}$ is

```
1  function largestRobustPart(C, F, N, t_max)
2    create a copy C' of C;
3    foreach component g ∈ C' do
4      replace g by g'[g, f_g, F];
5    done;
6    for t = 1...t_max do
7      unroll C' and C for t cycles;
8      force at least one pair of POs to different values;
9      convert to SAT instance;
10     for k = 1...N do
11       constrain Σ f_g = k;
12       while (satisfiable) do
13         G = {g | f_g == 1};
14         T := T ∪ G;
15         add constraint ⋁_{g∈G}(f_g == 0);
16       done;
17     done;
18   done;
19   S := C \ T;
20   return S;
21 end function;
```

Fig. 1. Algorithm to determine robustness

created (line 2). As shown in Figure 2 a fault predicate f_g is associated with each component $g \in \mathcal{C}'$ (lines 3-4). If $f_g == 1$, the function of g is modified; otherwise g behaves as in the fault free case. In the next step, the sequential equivalence check of $\mathcal{C}'$ and $\mathcal{C}$ is performed. For this purpose both circuits are "unrolled" for t time steps (line 7). The fault predicate of each component remains the same for all time steps. Moreover, a difference at least at one pair of primary outputs of the two circuits is enforced (line 8). The result is illustrated in Figure 3. The problem instance created by this algorithm is only satisfiable if the modification of a component causes different output responses of the circuits. If in-equivalence cannot be shown, the number of time steps considered is increased up to $t_{\max}$ (line 6). To guarantee that all components are calculated the modification of which causes faulty behavior, $t_{\max}$ has to be at least equal to the maximal sequential depth of a product automaton of $\mathcal{C}$ and $\hat{\mathcal{C}} \in \mathbb{C}_{\mathcal{C},\mathcal{S},\mathcal{F},N}$.

Now, by calculating all satisfying assignments (lines 10-17), all components are determined that cause faulty behavior when modified according to $\mathcal{F}_N$. Additionally, the number of fault predicates set to 1 is restricted to at most k (line 11) and iteratively incremented to N (line 10) to calculate all non-robust subcircuits up to N-fold faults. These non-robust components are joined into the set $\mathcal{T}$ (line 14). The complement set of $\mathcal{T}$ with respect to $\mathcal{C}$ yields the set $\mathcal{S}$ of Definition 3 (line 20).

The algorithm presented so far is restricted to the fault model $\mathcal{F}_N$. This results from the modification of a component g as shown in Figure 2. In the

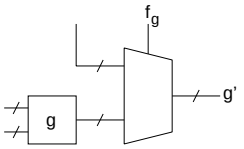

Fig. 2. Modification of a component

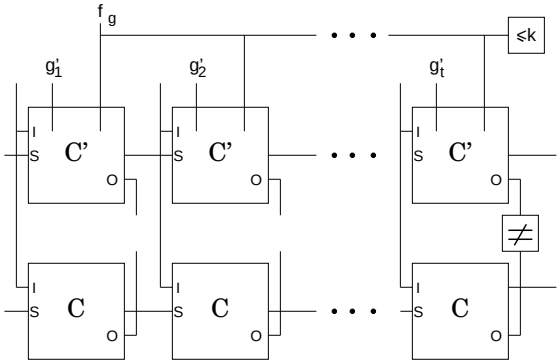

Fig. 3. SAT instance

faulty case $f_g == 1$, the component g may behave non-deterministically like a primary input. For fault models $\mathcal{F}_C$ and $\mathcal{F}_L$ additional constraints are necessary that force g to behave deterministically.

For fault model $\mathcal{F}_C$ this means in more detail: If the assignment of state bits and primary inputs in time step t is equal to that in time step t' then the output value of g has to be identical in both time steps.

For fault model $\mathcal{F}_L$ deterministic behavior is only required with respect to the direct predecessors of g.

The following theorem is the result.

Theorem 2. *A circuit $\mathcal{C}$, a fault model $\mathcal{F}$ and a positive integer N are given. Furthermore let $\mathcal{S}$:=*largestRobustPart($\mathcal{C}$, $\mathcal{F}$, N, $t_{\max}$)*. The circuit $\mathcal{C}$ has a robustness of $R_{\mathcal{F},N} = \frac{|\mathcal{S}|}{|\mathcal{C}|}$ if $t_{\max}$ is larger or equal to the sequential depth of the product automaton of $\mathcal{C}$ and $\mathcal{C}'$.*

5 Discussion

Sequential equivalence checking needs a large amount of resources regarding time and memory. Therefore several methods are presented in this section to improve the efficiency of the calculation.

5.1 Reduction to Property Checking

The functionality of a circuit can be exploited to significantly reduce the complexity of the calculation of robustness. Often a fault tolerant circuit includes logic to signal the occurrence of an internal malfunction. After discovering the first malfunction, either external actions can be taken to return the circuit into a fault free state (e.g. by restarting the system in case of transient faults) or the circuit is replaced (in case of static faults). This functionality can be instrumented to prevent the need for sequential equivalence checking. Instead, an inductive proof is applied that consists of multiple formal properties. Each individual property only argues over a few cycles. The base of this proof is an invariant that describes the fault free state of the system. The robustness of the circuit is then calculated with almost the same algorithm as introduced above. Instead of the fault free circuit, the property is used as the reference to model correct behavior. The only disadvantage of this approach is that it is not fully automatic. The properties and, especially, the invariant (to avoid reachability analysis) have to be determined manually for each circuit.

The inductive proof is structured as follows:

1. **Precondition:**
   Starting from the initial state, the system state is captured by an invariant *Inv* in the fault free case.
2. **Step:**
   The assumption is that no fault occurred so far, i.e. the invariant *Inv* is valid. Then, a case split is done for the fault free and the faulty case.
   (a) There occurs no fault.
   A property proves that the circuit transitions from a fault free state into another fault free state and that the logic for fault detection does not signal a malfunction, i.e. the invariant *Inv* is verified.
   (b) A fault occurs.
   A property proves that a transition into a state that is unreachable if no fault occurs is recognized by the fault detection logic, i.e. if the invariant *Inv* becomes invalid, the occurrence of a fault is signaled.

The precondition and case (a) of the induction step are proven by a traditional property checker. Only step 2.(b) requires the modeling technique presented in Section 4.

The inductive proof prevents reachability analysis for faulty circuits. Only the proof has to be carried out that any transition into an – in the fault free case – unreachable state is detected. The number of time steps that have to be considered depends on the functionality of the fault detection logic. In the simplest case, each occurrence of an unreachable state is detected immediately. Then the consideration of a single time step is sufficient.

5.2 Improving the Efficiency

The algorithm presented so far is complete but the complexity of the sequential equivalence check or the property check under fault assumptions is quite high. Therefore methods are proposed in the following to improve the efficiency.

Analogously to automatic test pattern generation or formal verification other engines besides a SAT prover can be assembled to solve the problem. The simulation of random stimuli and fault simulation can be applied to determine those components that may cause a deviation from the specification. Such components do not have to be handled afterward, i.e. no fault predicates f_g have to be assigned to these components since they are already classified as being non-robust. This reduces the search space. Moreover, this way an upper bound for the robustness of the circuit is determined because some of the components that are non-robust are identified – but not all of them.

Additionally, a combinational equivalence check can be applied to rule out those components that definitely cannot cause a deviation from the specification. For this purpose a combinational equivalence check is applied instead of the sequential one. Components that can be modified without changing the state transition function or the output response in this case, do not have to be considered in the sequential equivalence check any more. As a result a – often coarse – lower bound for the robustness is calculated.

Another improvement in efficiency can be achieved by exploiting the structure of the circuit. Initially, faults are only injected into state bits. Only if a modification of a state bit may cause an incorrect output response the preceding combinational logic has to be considered at all. Moreover, in this case the combinational logic only has to be modified in a way to reach the faulty state that was determined previously – the propagation of the fault does not have to be considered any more. In a similar way as proposed in [18] the hierarchical structure of the circuit can be exploited to analyze modification of coarse modules at first and only consider the fine grain structure of those modules that are not robust.

Finally, instead of calculating the exact robustness, the determination of an upper bound for $R_{\mathcal{F},N}$ is possible. For this purpose $t_{\max}$ is set to a smaller value than the sequential depth of $\mathcal{C}$ and $\hat{\mathcal{C}} \in \mathbb{C}_{\mathcal{C},\mathcal{S},\mathcal{F},N}$. In practice this works for most cases.

6 Experimental Results

In the following several robust and non-robust circuits are considered. All experiments are carried out with respect to the fault model $\mathcal{F}_N$. All run times are measured on an AMD Athlon 64 3500+ with 1GB running Linux.

Results for the reduction to sequential equivalence checking are presented in Table 1. The $(\mathcal{F}_N, 1)$-robustness of the circuits is determined. For this purpose $t_{\max}$ was not determined analytically. Instead the fixed values 5, 10 and 15 were considered. The influence of $t_{\max}$ on the run time is shown for one example. Besides the value of $t_{\max}$ the table shows the number of components (#comp), the number of state bits (#FF) and gates (#gt) in the complete problem instance. Furthermore the number of components in the faulty circuit $\mathcal{C}'$ ($|\mathcal{C}'|$) as well as run times in CPU seconds for a "standard" sequential equivalence check (sec)

Table 1. Run times for sequential equivalence checking for $N = 1$

		total			faulty			
$\mathcal{C}$	$t_{\max}$	#comp	#FF	#gt	$\lvert\mathcal{C}'\rvert$	sec	rsec	$R_{\mathcal{F}_N,1}$
s1269	5	624	74	1043	308	27,0s	88,4s	5%
r_s1269	5	1948	244	6514	970	14,8s	19185,3s	98%
rCounter	5	146	122	1505	70	0,2s	2,8s	97%
rCounter	10	146	122	1505	70	2,2s	21,7s	97%
rCounter	15	146	122	1505	70	13,3s	195,7s	97%

Table 2. Run times for the inductive approach

					step		
$\mathcal{C}$	$\lvert\mathcal{C}\rvert$	#FF	#gt	prec.	case (a)	case (b)	$R_{\mathcal{F}_N,1}$
rCounter	79	25	370	<0,1s	<0,1s	0,2s	100%

and the calculation of robustness using sequential equivalence checking (rsec) are given. The robustness ($R_{\mathcal{F}_N,N}$) of the circuit is shown in the last column.

As can be expected the ISCAS89 benchmark circuit *s1269* is not very robust yielding a robustness value of 5%. The second variant *r_s1269* of the circuit using *Triple Modular Redundancy* (TMR) is significantly more robust. The output values are determined by taking the majority of three instances of the circuit. Only faults in non-redundant parts of the circuit (e.g. the reset logic) may cause incorrect behavior. As a result a robustness of 98% is achieved.

The circuit *rCounter* is a counter with TMR, again three counters are instantiated and the majority determines the output value. Instead of gates, expressions on the RT-level were considered as components for this circuit. If the internal value of one instance deviates, a fault is signaled. Therefore a deviation from the specification is detected immediately for single faults. As a result the circuit is $(\mathcal{F}_N, 1)$-robust. Again, some parts of the circuit are not redundant. Therefore the robustness is below 100%.

In comparison to the standard sequential equivalence check, the calculation of robustness is significantly more time consuming. This is due to the larger number of primary inputs that yields a large search space. Especially, the example *rCounter* shows that increasing the value of $t_{\max}$ causes a drastic increase in run time. In particular the maximal sequential depth of the product automaton of fault free circuit and faulty circuit cannot be met to determine the exact robustness. For this purpose an improvement of the efficiency of the technique is necessary.

In case of *rCounter* this can be done by reducing the problem to property checking and exploiting the fault detection logic. The experimental results are shown in Table 2. The size of the circuit is slightly increased (79 instead of 70 components) because now the logic for fault detection is also considered. Besides the data given above already, the run times for the three parts of the inductive proof are shown. Only a single time step had to be considered, as any deviation of the internal states of the three counters is detected. As a result a drastic

reduction of the run time is achieved. Even the sum of the run times for all three steps is clearly below one second. A robustness of 100% is achieved because a fault in the reset state is not modeled. Due to single faults either the output value is correct or the logic for fault detection functions correctly.

Overall measuring robustness by implicitly enumerating all faults is possible. A significant improvement of the efficiency is achieved by reducing the problem to a sequence of property checking instances.

7 Summary

An approach to automatically calculate the robustness of circuits was proposed. A fully automatic method can be established when reducing the problem to sequential equivalence checking. The run time to calculate the robustness is significant in this case. Therefore methods to improve the efficiency have been proposed. In particular, the problem can be reduced to property checking. The method is only semi-automatic in this case, because the corresponding properties have to be created manually. But as an advantage a drastic reduction of the run times is achieved.

Future work involves further improvements in efficiency on the algorithmic level and a thorough investigation of the other fault models.

References

1. Miskov-Zivanov, M., Marculescu, D.: Circuit reliability analysis using symbolic techniques. IEEE Trans. on CAD **25** (2006) 2638–2649
2. Civera, P., Macchiarulo, L., Rebaudengo, M., Reorda, M.S., Violante, M.: An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits. Jour. of Electronic Testing: Theory and Applications **18** (2002) 261–271
3. Leveugle, R.: A new approach for early dependability evaluation based on formal property checking and controlled mutations. In: IEEE International On-Line Testing Symposium. (2005) 260–265
4. Krautz, U., Pflanz, M., Jacobi, C., Tast, H.W., Weber, K., Vierhaus, H.T.: Evaluating coverage of error detection logic for soft errors using formal methods. In: Design, Automation and Test in Europe. (2006) 176–181
5. Zhou, Q., Mohanram, K.: Gate sizing to radiation harden combinational logic. IEEE Trans. on CAD **25** (2006) 155–166
6. Smith, A., Veneris, A., Ali, M., Viglas, A.: Fault diagnosis and logic debugging using boolean satisfiability. IEEE Trans. on CAD **24** (2005) 1606–1621
7. Pixley, C.: A theory and implementation of sequential hardware equivalence. IEEE Trans. on CAD **11** (1992) 1469–1478
8. Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp. **35** (1986) 677–691
9. Davis, M., Logeman, G., Loveland, D.: A machine program for theorem proving. Comm. of the ACM **5** (1962) 394–397
10. Eén, N., Sörensson, N.: An extensible SAT solver. In: SAT 2003. Volume 2919 of LNCS. (2004) 502–518

11. Tseitin, G.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, Part 2. (1968) 115–125 (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).
12. Larrabee, T.: Test pattern generation using Boolean satisfiability. IEEE Trans. on CAD **11** (1992) 4–15
13. Cook, S.: The complexity of theorem proving procedures. In: 3. ACM Symposium on Theory of Computing. (1971) 151–158
14. Marques-Silva, J., Sakallah, K.: Conflict analysis in search algorithms for propositional satisfiability. In: IEEE International Conference on Tools with Artificial Intelligence. (1996)
15. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Design Automation Conf. (2001) 530–535
16. Smith, A., Veneris, A., Viglas, A.: Design diagnosis using Boolean satisfiability. In: ASP Design Automation Conf. (2004) 218–223
17. Staber, S., Fey, G., Bloem, R., Drechsler, R.: Automatic fault localization for property checking. In: IBM Haifa Verification Conference. Volume 4383 of LNCS., Springer Verlag (2006)
18. Ali, M., Safarpour, S., Veneris, A., Abadir, M., Drechsler, R.: Post-verification debugging of hierarchical designs. In: Int'l Conf. on CAD. (2005) 871–876