Robustness Check for Multiple Faults Using Formal Techniques

Stefan Frehse, Görschwin Fey, André Sülflow, and Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{sfrehse,fey,suelflow,drechsle}@informatik.uni-bremen.de

Abstract. Feature sizes in VLSI circuits are steadily shrinking. This results in increasing susceptibility to soft errors, e.g. due to environmental radiation. Precautions against soft errors can be taken on all design stages, e.g. the architectural level, algorithmic level, or on the layout level. Whether the final implementation contains flaws or really provides robustness to soft errors remains to be checked.
Here, we propose an approach to formally verify the robustness of a circuit with respect to multiple soft errors. We propose a fault model that prunes the exponentially sized space of multiple soft errors and an algorithm that automatically analyzes a given circuit.

1 Introduction

Moore's law is still valid due to continuously shrinking the feature sizes in VLSI circuits. The small feature size allows for low-power circuitry operating at high frequencies and for assembling millions of components on a chip. On the other hand less energy is required to drive smaller transistors. Consequently, the soft error rate increases. In the future these issues have to be considered during chip design [1].

Upsets or bit-flips caused by environmental radiation are one source of soft errors. A *Single Event Upset* (SEU) causes a single bit-flip. Multiple SEUs distributed in time may accumulate in the state of the circuit or a single particle with high energy may cause multiple soft errors in spatially close components of the circuit. This is denoted as *Multiple Event Upset* (MEU) in the following. The number of MEUs is exponential in the number of time frames considered and the number of components of the considered circuit.

Numerous techniques at all design stages are available to catch soft errors before they manifest in the output response or in the state of a circuit: error correcting codes [2] have been proposed, redundancy in time [3] or space [4] is applied, circuit structures are widened and thereby hardened during layout [5,6]. But when implementing these techniques in a design, bugs may be introduced. Therefore, like the functionality, also the robustness of an implementation has to be verified.

Simulation-based and emulation-based techniques [7, 8] to validate this kind of robustness have been proposed. But even for small circuits these approaches can only cover a small portion of the input space and state space even for single errors. Considering MEUs further decreases the coverage achieved.

Formal approaches cover all input stimuli, any state of a design and any fault. First approaches have been presented in [9–12]. The work in [10] requires manual interaction. The techniques in [9, 10, 12] are restricted to single errors. The approach in [12] provides a measure for robustness. Even if the formal approach cannot finish, bounds on robustness are returned. Only the algorithm presented in [11] covers MEUs, but does not propose any run time improvements to handle the huge space of potential faults.

Here we propose an approach to formally verify robustness with respect to MEUs. Modeling upsets is similar to [12] and the bounds on robustness are extended to MEUs. The main contributions of our work are:

– A *fault model* for MEUs that prunes the search space,
– a *measure for robustness* with respect to MEUs,
– an *algorithm* to automatically verify robustness, and
– *bounds* on robustness while the algorithm proceeds.

Our algorithm runs fully automatically and can therefore be seamlessly integrated into the design flow, serving as a push-button tool. Experiments on non-robust benchmarks from the ITC'99 benchmark set and on derived robust circuits show the effectiveness of our technique.

The paper is structured as follows: Section 2 reviews preliminaries. The fault model for MEUs is motivated and described in Section 3. Section 4 explains the algorithm, the robustness measure and the bounds. Experimental results are reported in Section 5. Finally, conclusions are stated in Section 6.

2 Preliminaries

2.1 Boolean Satisfiability – SAT

The *Boolean Satisfiability* (SAT) problem is a decision problem that asks whether there exists a variable assignment of a Boolean formula $f : \mathbb{B}^n \rightarrow \mathbb{B}$ such that the formula evaluates to one. If such an assignment exists the formula is called *satisfiable*, otherwise *unsatisfiable*. The SAT problem is in the class of the $\mathcal{NP}$-complete problems, proved by Cook in 1971 [13]. Problem instances coming from practical problems [14] can often be solved effectively by state-of-the-art SAT solvers [15, 16]. The most common form of Boolean formulas for SAT solvers is the *Conjunctive Normal Form* (CNF).

2.2 Sequential Circuit Model

We consider a synchronous sequential circuit $\mathcal{C}$ with *Primary Inputs* PI($\mathcal{C}$), *Primary Outputs* PO($\mathcal{C}$) and *State elements* S($\mathcal{C}$). The number of components of

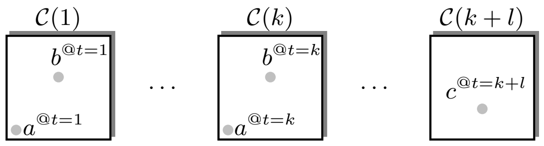

Fig. 1. Multiple faults in a sequential circuit

the circuit $\mathcal{C}$ is denoted by $|\mathcal{C}|$. Here, a component may be a gate, module or a source level expression in a hardware description language.

A circuit can be converted in linear time and space into a CNF with respect to the circuit size [17].

3 Fault Model

This section first discusses MEUs and our notion of robustness. Then the fault model and further properties to prune the search space are introduced.

3.1 Multiple Event Upsets and Robustness

In the following our notion of MEUs is described. A single upset on the logical level is modeled as proposed in [12] by non-deterministically changing the values of an internal wire.

A MEU is composed of multiple SEUs. Relevant data to uniquely describe a MEU is the following: at which components of the circuit the upsets occur and at which point in time the value of a component is changed.

Example 1. For example some MEUs of a sequential circuit are shown in Figure 1: $\alpha_1 = (a^{@t=1}, a^{@t=k})$, $\alpha_2 = (a^{@t=1}, b^{@t=k})$, $\alpha_3 = (b^{@t=1}, a^{@t=k})$ and $\alpha_4 = (a^{@t=1}, b^{@t=k}, c^{@t=k+l})$.

Any algorithm that evaluates the robustness of a circuit with respect to MEUs has to adequately model which components are affected at which point in time. For complexity reasons we restrict the observation time to a window of $t_{\max}$ time frames. This restriction is required for complexity reasons but is also justified by practical assumptions. A MEU should be detected within a short period of time signaled by a fault detection signal `flt`, cause *Silent Data Corruption* (SDC) or disappear. In more detail there are three alternatives after a MEU occurred in a circuit:

1. The effect propagates to the outputs within $t_{\max}$ time frames, causing incorrect output behavior. The circuit is *non-robust* with respect to this MEU.
2. The MEU may manifest in the state, is not detected, and remains hidden in the system. The effect may be observable at the outputs at a later point in time. The circuit is *non-classified* with respect to this MEU – which corresponds to a SDC.

3. The effect disappears or is recognized by fault detection logic. The circuit is *robust* with respect to this MEU.

In the following we consider a MEU as being *non-robust*, *non-classified* or *robust*, respectively.

3.2 Fault Model

To simplify the representation our fault model abstracts from the points in time, this is modeled by *Abstracted MEUs* (AMEUs). Moreover, we abstract from the order of the single events composing a MEU by defining an equivalence relation on AMEUs. Thus, multiple MEUs are mapped to a single AMEU.

Assume a MEU is found to be non-robust, computed by solving a SAT-Problem. This MEU is mapped to an AMEU that is also determined non-robust. Additionally, all equivalent AMEUs are also considered non-robust without further search. By this, the set of non-robust MEUs is over-approximated. For diagnosis and to improve the fault tolerance, the AMEU can be enriched with time information to reconstruct the complete scenario for better understanding.

The abstraction by AMEUs prunes the search space that has to be explored by the algorithm to prove robustness.

To model AMEUs, a single component $g \in \mathcal{C}$ has to be represented multiple times to model multiple faults of a single component.

Given a circuit $\mathcal{C}$ and a component $g \in \mathcal{C}$. To represent g multiple times in a set, g is marked with a superscript k denoted by $g^{(k)}$. The set

$$M^{\mathcal{C}}(k) = \{g^{(k)} | g \in \mathcal{C}\}$$

marks all components of $\mathcal{C}$ with the superscript k.

In the following $\eta \in \mathbb{N}$ specifies the number of flipped bits caused by a MEU or an AMEU, repsectively. The parameter η is called *fault cardinality.*

Definition 1. *Let $\mathcal{C}$ be a circuit and $\eta \in \mathbb{N}$ the fault cardinality. The set*

$$\mathbb{F}^{\mathcal{C}}_{\eta} = \bigcup_{i=1}^{\eta} \{\alpha | \alpha \subseteq \bigcup_{k=1}^{\eta} M^{\mathcal{C}}(k), |\alpha| = i\}$$

contains all possible AMEUs up to the fault cardinality η. The set is called AMEU-set*.*

The cardinality of the AMEU-set is given by

$$|\mathbb{F}^{\mathcal{C}}_{\eta}| = \sum_{i=1}^{\eta} \binom{\eta \cdot |\mathcal{C}|}{i}$$

Instead of explicitly defining the mapping of MEUs to AMEUs we give some examples in the following.

Example 2. The MEUs of Example 1 are mapped to AMEUs as shown in Table 1.

Table 1. Mapping MEUs to AMEUs

MEU	AMEU
$(a^{@t=1}, a^{@t=k})$	$\{a^{(1)}, a^{(2)}\}$
$(a^{@t=1}, b^{@t=k})$	$\{a^{(1)}, b^{(1)}\}$
$(b^{@t=1}, a^{@t=k})$	$\{a^{(1)}, b^{(1)}\}$
$(a^{@t=1}, b^{@t=k}, c^{@t=k+l})$	$\{a^{(1)}, b^{(1)}, c^{(1)}\}$

Example 3. Let $\mathcal{C}_{\text{ex}} = \{a, b, c\}$ be a circuit with three components and $\eta = 2$ faults are considered. The sets $M^{\mathcal{C}_{\text{ex}}}(1) = \{a^{(1)}, b^{(1)}, c^{(1)}\}$ and $M^{\mathcal{C}_{\text{ex}}}(2) = \{a^{(2)}, b^{(2)}, c^{(2)}\}$ mark the components to get a representation of a single component for multiple time frames. The set of all faults is given by:

$$\begin{aligned}
\mathbb{F}_2^{\mathcal{C}_{\text{ex}}} &= \{\{a^{(1)}\}, \{b^{(1)}\}, \{c^{(1)}\}, \{a^{(2)}\}, \{b^{(2)}\}, \{c^{(2)}\}\} \\
&\cup \{\{c^{(1)}, b^{(2)}\}, \{a^{(1)}, c^{(2)}\}, \{a^{(1)}, b^{(1)}\}, \{c^{(1)}, a^{(2)}\}\} \\
&\cup \{\{c^{(1)}, c^{(2)}\}, \{a^{(2)}, b^{(2)}\}, \{a^{(1)}, c^{(1)}\}, \{b^{(1)}, c^{(1)}\}\} \\
&\cup \{\{b^{(1)}, a^{(2)}\}, \{b^{(2)}, c^{(2)}\}, \{a^{(1)}, b^{(2)}\}, \{b^{(1)}, c^{(2)}\}\} \\
&\cup \{\{a^{(2)}, c^{(2)}\}, \{a^{(1)}, a^{(2)}\}, \{b^{(1)}, b^{(2)}\}\}
\end{aligned}$$

The superscript should only reflect the number of times a certain component was involved in a MEU. An AMEU $\{a^{(1)}, a^{(2)}\}$ is of interest, because component a has been hit two times by a MEU. But no MEU is mapped to the AMEU $\{a^{(1)}, a^{(3)}\}$ or to $\{a^{(2)}, b^{(3)}\}$. Instead $\{a^{(1)}, a^{(2)}\}$ and $\{a^{(1)}, b^{(1)}\}$ will be used, respectively. Here we define an equivalence relation to reduce the number of AMEUs to be considered. Distinct but equivalent AMEUs do not necessarily cause the same errors. The designer still knows the components that are non-robust.

Two AMEUs β and $\tilde{\beta}$ are considered equivalent, iff the number of occurrences of each component in β and $\tilde{\beta}$ is equal. This can formally be defined as an equivalence relation $\sim$. Let

$$\beta \sim \tilde{\beta} = \{(\beta, \tilde{\beta}) \in \mathbb{F}_\eta^{\mathcal{C}} \times \mathbb{F}_\eta^{\mathcal{C}} |\ \forall g \in \mathcal{C} : \text{cnt}(\beta, g) = \text{cnt}(\tilde{\beta}, g)\}$$

be the relation on two equivalent AMEUs, whereas β is an AMEU and g is a component of a circuit. The function cnt counts the occurrences of a component in an AMEU:

$$\begin{aligned}
\text{cnt}(\{a^{(i)}\} \cup \beta, g) &= \text{cnt}(\beta, g) + \begin{cases} 1 & \text{if } g = a \\ 0 & \text{otherwise} \end{cases} \\
\text{cnt}(\varnothing, g) &= 0
\end{aligned}$$

The relation $\sim$ is reflexive, symmetric and transitive. The equivalence class $[\beta]_\sim$ of an AMEU β contains all equivalent AMEUs with respect to $\sim$. All AMEUs in

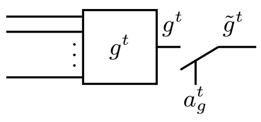

Fig. 2. Fault injection

an equivalence class are considered non-robust, if one member is non-robust. This is valid in terms of our conservative approach. If one member is non-robust, then all the other members are also potentially vulnerable for non-robustness. This equivalence relation speeds up the classification and AMEUs can be classified non-robust without finding a corresponding non-robust MEU.

An additional property of AMEUs further speeds up the classification of non-robust and non-classified AMEUs. Adding another single fault to a non-robust MEU causes faulty behavior, independent of the value of the fault injection. This is extended to AMEUs. If an AMEU β is non-robust, then all AMEUs which include β are also non-robust:

$$\forall \gamma \in \mathbb{F}_\eta^{\mathcal{C}} : \beta \subset \gamma \Rightarrow \gamma \quad \text{is non-robust}$$

In general, the *implication-set* contains all AMEUs which result from one AMEU.

Definition 2. *Given a circuit $\mathcal{C}$, the fault cardinality η and a subset $M \subseteq \mathbb{F}_\eta^{\mathcal{C}}$. Furthermore let $\beta \in \mathbb{F}_\eta^{\mathcal{C}}$ be an AMEU, then the set*

$$\mathbb{I}^{\text{impl}}(M, \beta) = \{\gamma | \gamma \in M, \beta \subseteq \gamma\}$$

represents all AMEUs which include β. The set is called implication-set.

Example 4. Consider Example 3 and let $\beta = \{a^{(1)}\}$ be a non-robust AMEU. The implication-set for β results in:

$$\begin{aligned}\mathbb{I}^{\text{impl}}(\mathbb{F}_\eta^{\mathcal{C}}, \beta) = \{&\{a^{(1)}\}, \{a^{(1)}, c^{(2)}\}, \{a^{(1)}, b^{(1)}\}\} \\ &\cup \{\{a^{(1)}, c^{(1)}\}, \{a^{(1)}, b^{(2)}\}, \{a^{(1)}, a^{(2)}\}\}\end{aligned}$$

Here, five AMEUs are classified as non-robust additionally, since β is non-robust. Moreover, fault equivalence is exploited. If $\beta = \{a^{(1)}\}$ is non-robust then $\gamma = \{a^{(2)}\}$ is also non-robust. Consequently, the implication-set of γ results in five additional classifications. In total during the classification of β ten additional classifications are computed. These implications reduce the complexity and save run time during the classification.

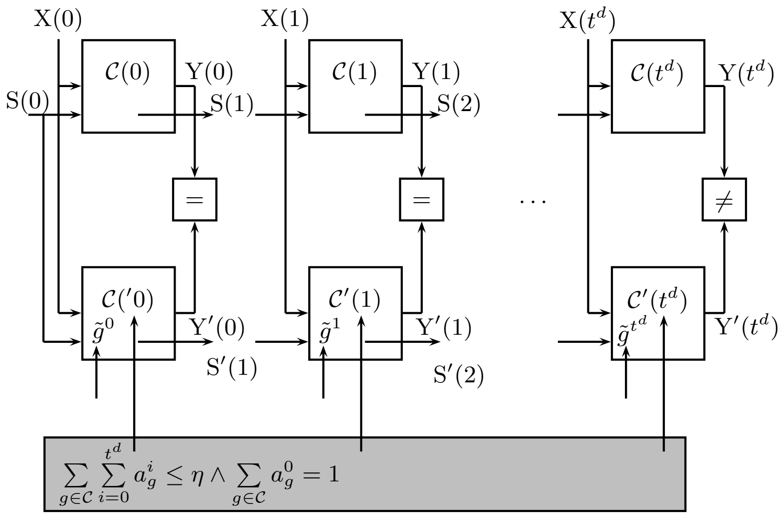

Fig. 3. Sequential Model

4 Algorithm

This section introduces the algorithm to evaluate the robustness of a circuit. The algorithm uses a SAT engine to determine all non-robust MEUs and non-classified MEUs. These are mapped to AMEUs to keep track of the search space already explored. Essentially, a *Sequential Equivalence Check* (SEC) of the original circuit compared to the circuit after injecting a MEU is performed.

4.1 Overview

Injection of an upset at a single component is shown in Figure 2. The output signal of a component $g \in \mathcal{C}$ is associated to variable g^t at time frame t. For a component g^t at time frame t fault injection logic is inserted. A *fault predicate* a_g^t and a new variable $\tilde{g}^t$ are introduced. The output g^t is replaced by $\overline{a}_g^t \Rightarrow (\tilde{g}^t = g^t)$. If the fault predicate is off, i.e. $a_g^t = 0$, then the component behaves normally. Otherwise if the fault predicate is activated, i.e. $a_g^t = 1$, any value can be injected in the circuit.

To determine non-robust MEUs, a SAT instance is created as shown in Figure 3. Similar to bounded model checking the circuit $\mathcal{C}(t)$ and $\mathcal{C}'(t)$ are unrolled for t^d time frames. For every unrolled time frame the PIs of both circuits are connected. The initial state $S(0)$ of both circuits is constrained as equal. By allowing any *reachable* state for $S(0)$ the algorithm remains complete. The POs at time frame t^d are forced to be different. Finally, a cardinality constraint as shown in Figure 3 is added to the SAT instance to constrain that less than $\eta + 1$ components are modified and at least one single fault occurs in the first time frame. The injection of at least one fault in the first time step, further shrinks the search space. The algorithm remains complete, because any reachable state is considered at $S(0)$. The reachable states can be computed by e.g. reachability analysis based on BDDs [18].

The SAT instance is satisfied iff a set of fault predicates $P = \{a_g^{k_1}, a_h^{k_2}, \ldots, a_j^{k_l}\}$ with $|P| \leq \eta$ are activated and the injected values lead to a difference at the primary outputs. The MEU $(g^{@t=k_1}, h^{@t=k_2}, \ldots, j^{@t=k_l})$ represented by fault predicates is classified as non-robust and the corresponding AMEU is added to the set of non-robust AMEUs. The fault predicates of α are blocked in the SAT instance to get all non-robust AMEUs by computing all satisfying solutions. If no more MEUs can be classified non-robust within the time bound t^d, t^d is incremented until $t_{\max}$ is reached. The circuit $\mathcal{C}(t)$ and the copy $\mathcal{C}'(t)$ are appended to the existing model. The computation of the non-robust MEUs is repeated.

After reaching $t_{\max}$, non-classified AMEUs are determined analogously. Here, differing states and `flt` $= 0$ from time frame 0 to time frame $t_{\max}$ are constrained.

4.2 Pseudo Code

The pseudo code is shown in Algorithm 1. Parameters are the circuit $\mathcal{C}$, the fault cardinality η and the bound $t_{\max}$ for the maximum number of time frames considered.

The circuit $\mathcal{C}$ and a copy $\mathcal{C}'$ are unrolled for $t \in [0, \ldots, t_{\max}]$ time steps (line 4). At least one fault predicate is activated in the first time step (line 7). The initial states of $\mathcal{C}$ and $\mathcal{C}'$ are constrained to be equal (line 8). For each component the fault injection logic, described above is inserted (line 10–12). To reduce the number of blocking clauses when searching for all satisfying solutions, an extended fault injection logic is used. A fault predicate is extended by two new variables, w_g^t and w_g. The first variable is created for each time frame and the second for all time frames. The fault predicate a_g^t is the conjunction of w_g^t and w_g. By this, a component can be blocked for all time frames, by blocking the variable w_g to evaluate to true. Due to this construction only one blocking clause is required for all permutations of a set of components.

The miter circuit for SEC is created (line 13). Now the algorithm determines the non-robust MEUs. The constraint NR is created to determine all MEUs which cause a difference at the POs, while the fault signal `flt` does not detect a fault. The solutions are extracted by the method extractAllSolutions and afterwards the constraint NR is removed. The extracted non-robust AMEUs are added to the set $\mathbb{S}$ (line 18). Then, the comparison logic of the miter circuit is removed and a new iteration with an incremented time frame is started.

After unrolling the circuit $t_{\max}$ times the non-classified MEUs are determined analogously to the non-robust MEUs (line 22–26). Finally, the robust fault candidates can be computed as shown in line 27. The sets of robust, non-robust and non-classified AMEUs are returned.

The extraction of non-robust and non-classified AMEUs is done by the method extractAllSolutions in Algorithm 2. This ethod receives the fault cardinality (η), the set of non-classified AMEUs $(\mathbb{U})$ and returns the extracted

Algorithm 1: computeRobustness

```
Input: $\mathcal{C}$ the circuit, $\eta$ number of faults, $t_{\max}$ max time to unroll
Output: $(\mathbb{T}, \mathbb{S}, \mathbb{U})$ set of the robust, non-robust and non-classified AMEUs
1  begin
2      $\mathbb{T} = \mathbb{S} = \varnothing$;
3      for $t = 0$ to $t_{\max}$ do
4          create copies of $\mathcal{C}(t)$ and $\mathcal{C}'(t)$ of $\mathcal{C}$;
5          constraint $\mathrm{PI}(\mathcal{C}(t)) \Leftrightarrow \mathrm{PI}(\mathcal{C}'(t))$;
6          if $t = 0$ then
7              add constraint $\sum_{g \in \mathcal{C}} a_g^0 \geq 1$;
8              add constraint $\mathrm{S}(\mathcal{C}(t)) \Leftrightarrow \mathrm{S}(\mathcal{C}'(t))$;
9          end
10         foreach $g^t \in \mathcal{C}'(t)$ do
11             replace $g^t$ by ${g'}^t[g^t, a_g^t]$ and $a_g^t \Leftrightarrow w_g^t \wedge w_g$;
12         end
13         $\mathrm{cmp_{po}}$ = create miter of all $\mathrm{PO}(\mathcal{C}(t), \mathcal{C}'(t))$;
14         // compute non-robust AMEUs;
15         add constraint $\mathrm{NR} = \mathrm{cmp_{po}} \ \wedge \ \overline{\texttt{flt}} = 1$;
16         $\mathbb{S}' = $ extractAllSolutions$(\eta, \mathbb{F}_\eta^{\mathcal{C}})$);
17         remove constraint NR;
18         $\mathbb{S} = \mathbb{S} \cup \mathbb{S}'$;
19         remove $\mathrm{cmp_{po}}$;
20     end
21     // compute non-classified AMEUs;
22     $\mathrm{cmp_s}$ = create miter of all $\mathrm{S}(\mathcal{C}(t_{\max}), \mathcal{C}'(t_{\max}))$;
23     $\mathrm{cmp_{po}}$ = create miter of all $\mathrm{PO}(\mathcal{C}(t_{\max}), \mathcal{C}'(t_{\max}))$;
24     add constraint $\mathrm{NC} = \overline{\texttt{flt}} \ \wedge \ \overline{\mathrm{cmp_{po}}} \ \wedge \ \mathrm{cmp_s} = 1$;
25     $\mathbb{U} = $ extractAllSolutions$(\eta, \mathbb{F}_\eta^{\mathcal{C}} \setminus \mathbb{S})$;
26     remove constraint NC;
27     $\mathbb{T} = \mathbb{F}_\eta^{\mathcal{C}} \setminus \mathbb{U} \setminus \mathbb{S}$;
28     return $(\mathbb{T}, \mathbb{S}, \mathbb{U})$;
29 end
```

AMEUs (M). Each iteration increments the number of the injected faults (line 3). The number of injected aults is limited by a cardinality constraint in line 4. The while-loop (line 5) extracts all solutions. If a solution exists, the MEU α is extracted. The corresponding AMEU is extracted and all equivalent AMEUs are added to the set of solutions. The AMEUs are now classified and have to be removed from the set $\mathbb{U}'$ (line 13). Afterwards, a blocking clause is inserted to the SAT instance.

In our implementation the algorithm stores the sets of AMEUs by using *Binary Decision Diagrams* [18] to have a compact set representation and to efficiently manipulate sets.

Algorithm 2: EXTRACTALLSOLUTIONS

```
Input: $\eta$ – fault-card., $\mathbb{U}'$ – set of non-classified fault candidates
Output: $M$ – set of all solutions
begin
    $M = \varnothing$;
    for $k = 1$ to $\eta$ do
        constraint limit: $\sum_{g \in \mathcal{C}} \sum_{i=0}^{t^d} a_g^i = k$;
        while SOLVER.SOLVE() do
            $\alpha = \{g_{i_j}^{@t=i} \mid a_g^l = 1\} = \{g_{i_1}^{@t=1}, \ldots, g_{i_k}^{@t=t_k}\}$;
            $\beta = \mathrm{AMEU}(\alpha)$;
            $A = \varnothing$;
            foreach $\tilde{\beta} \in [\beta]_\sim$ do
                $A = A \cup \mathbb{I}^{\mathrm{impl}}(\mathbb{U}', \tilde{\beta})$;
            end
            $M = M \cup A$;
            $\mathbb{U}' = \mathbb{U}' \setminus A$;
            insert blocking clause for all $[\beta]_\sim$ by blocking $w_g$
        end
        remove constraint limit;
    end
    return $M$;
end
```

4.3 Robustness Measure

After determining the set of non-robust AMEUs $\mathbb{S}$, the set of non-classified AMEUs $\mathbb{U}$ and the set of robust AMEUs $\mathbb{T}$ with respect to a time bound t^d, a lower bound and an upper bound on the robustness of the circuit can be given by extending the bounds of [12] from SEUs to AMEUs. Robustness is considered as the ratio of the number of robust AMEUs to the total number of AMEUs:

$$R_{lb} = \frac{|\mathbb{T}|}{|\mathbb{F}_\eta^{\mathbb{C}}|} = 1 - \frac{|\mathbb{S}| + |\mathbb{U}|}{|\mathbb{F}_\eta^{\mathbb{C}}|}$$

$$R_{ub} = \frac{|\mathbb{U}| + |\mathbb{T}|}{|\mathbb{F}_\eta^{\mathbb{C}}|} = 1 - \frac{|\mathbb{S}|}{|\mathbb{F}_\eta^{\mathbb{C}}||}$$

These bounds can also be determined when the algorithm does not progress to time frame $t_{\max}$ within the allowed computational resources. Instead, intermediate results can be returned.

The bounds roughly indicate whether the fault detection and correction logic of a circuit works properly or the circuit is susceptible to faults. For improving the implementation, the designer can additional consider the components involved in non-robust or non-classified AMEUs.

5 Experimental results

In this section the experimental results are presented. The algorithms were implemented in C++. All experiments were carried out on an Intel Core2 Duo (2.0GHz, 4GB RAM, Linux). The SAT solver Zchaff [16] is used with incremental extension. The time out is set to 2·1500s, the first 1500s are given to the classification of the non-robust AMEUs and the second time out limits the classification of the non-classified AMEUs. Up to 10 time frames are considered.

If the classification of the non-classified faults exceeds the time out, the set of all non-classified AMEUs is incomplete. In this case only an *upper* limit on the lower bound can be given, i.e. there may be more non-classified AMEUs and the lower bound may decrease further. This is denoted by ^{a} for *aborted*.

Circuits without fault tolerance mechanism were taken from the ITC'99 benchmark suite. We modified the circuits and applied *Triple Modular Redundancy* (TMR) as fault tolerance mechanism. We denote TMR circuits with the suffix `-tmr`. Some circuits are equipped with fault detection logic and a fault signal (denoted by `flt`). We also created 5 and 9 modular redundant circuits named `b02-5mr` and `b01-9mr`, respectively. The initial states $S(0)$ are constrained to reachable states, determined by a BDD-based reachability analysis.

In Table 2 the results of the experiments are shown. The column CIRCUIT gives the name of the circuits. Columns $|\mathcal{C}|$, $|S|$ and η give the number of components, of state elements and the considered fault cardinality, respectively. The time frame reached within the given time out is shown in column t^d. If the time out is reached or the circuit is unrolled for 10 time frames, a number of non-classified AMEUs remains that is shown in column $|\mathbb{U}|$. The cardinality of the AMEU-set for every circuit and the fault cardinality are given in column $|\mathbb{F}_\eta^\mathcal{C}|$. The number of non-classified AMEUs and the size of the AMEU-set are rounded. The resulting lower (R_{lb}) and upper (R_{ub}) bound of the robustness are shown in the last two columns.

The standard benchmark circuits `b01`, `b02`, `b03` and `b06` can be fully classified by the presented approach. The robustness of these circuits is very low when double faults are considered.

The TMR-circuits with a fault signal are classified almost completely. The resulting robustness is consistent with the expectation.

For the standard TMR-circuits the robustness is getting lower with increasing fault cardinality. For double faults the maximal number of considered time frames was reached for `b01-tmr` and `b02-tmr`. With an increasing number of components and state elements (e.g. `b06-tmr`), the number of time frames reached becomes smaller and the gap of the resulting bounds is larger. For `b02-tmr` the determination of the non-classified AMEUs exceeds the time out. Still an *upper* limit on the lower bound can be given. A large gap between upper and lower bound indicates that often fault effects do not propagate to the outputs but cause SDC.

Furthermore, for the TMR circuits equipped with a fault signal a high robustness is computed for double faults. The fault detection logic detects and

Table 2. Bounds of the robustness for multiple faults

Circuit	$\lvert\mathcal{C}\rvert$	$\lvert\mathbb{S}\rvert$	η	t^d	$\lvert\mathbb{U}\rvert$	$\lvert\mathbb{F}^{\mathcal{C}}_{\eta}\rvert$	$R_{lb}\%$	$R_{ub}\%$
Circuits without fault tolerance mechanism								
b01	64	5	2	10	-	8.26×10^{03}	0.77	0.77
b02	32	4	2	10	-	2.35×10^{03}	1.43	1.43
b03	199	30	2	10	-	7.94×10^{04}	0.25	0.25
b06	73	9	2	10	-	1.07×10^{04}	0.68	0.68
TMR-Circuits								
b01-tmr	212	15	2	10	3.31×10^{03}	9.01×10^{04}	1.64	38.42
			3	3	1.38×10^{07}	4.28×10^{07}	0.33	32.46
			4	3	3.62×10^{09}	2.15×10^{10}	0.03	16.87
b02-tmr	112	12	2	10	8.62×10^{03}	2.52×10^{04}	1.37	35.59
			3	4	8.29×10^{05}	6.33×10^{06}	3.58[a]	13.40
			4	4	6.69×10^{07}	1.67×10^{09}	1.74[a]	5.13
b06-tmr	275	27	2	5	7.15×10^{04}	1.52×10^{05}	8.87	56.06
			3	3	3.04×10^{07}	9.35×10^{07}	2.52	34.97
			4	3	3.21×10^{10}	6.09×10^{10}	7.02	60.10
TMR-circuits with a fault signal								
b01-tmrflt	215	15	2	6	2.68×10^{02}	9.27×10^{04}	81.26	81.55
b02-tmrflt	123	12	2	10	1.01×10^{03}	3.04×10^{04}	84.67	87.99
b06-tmrflt	286	27	2	4	1.22×10^{03}	1.64×10^{05}	72.50	73.24
Multiple redundancy circuits								
b02-5mr	188	20	2	10	6.22×10^{04}	7.09×10^{04}	2.04	89.81
			3	3	2.51×10^{07}	2.99×10^{07}	0.30	84.30
			4	3	9.86×10^{09}	1.32×10^{10}	0.00	74.30
b01-9mr	656	45	2	3	7.98×10^{05}	8.61×10^{05}	5.98[a]	98.73

[a] aborted during the classification of non-classified AMEUs.

signals faults within one time frame. This also holds for faults manipulating the state of the circuit causing SDC.

Consequently, the bounds are very close despite the small unrolling depth, i.e. a precise result about the robustness of the circuits is determined.

For multiple redundancy circuits without a fault signal, the gap of the bounds is relatively large, because of the small reached unroll time frame within the given computational resources. An *upper* limit on the lower bound for `b01-9mr` is determined.

In summary, despite the exponentially sized set of MEUs and the computational complexity of the sequential equivalence check, the exact robustness computation is done for all small circuits. Even for *larger* circuits bounds on the robustness are determined by the algorithm.

6 Conclusion

In this paper we proposed a fully automatic approach to determine the robustness of a sequential circuit for MEUs. The introduced fault model prunes the search space. To determine non-robust components a SEC is performed. The presented approach fully classifies the small ITC'99 circuits and provides bounds on robustness, when a full classification cannot be achieved within given resource limits.

For future work, we want to consider layout information to further prune the search space. A decision heuristic for SAT solvers adapted for the robustness check promises better run times. Furthermore other fault mechanisms for benchmarks will be considered.

References

1. Borkar, S.: Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. IEEE Micro **25**(6) (2005) 10–16
2. Hamming, R.W.: Error detecting and error correcting codes. Bell System Technical Jour. **9** (April 1950) 147–160
3. Austin, T., Bertacco, V.: Deployment of better than worst-case design: Solutions and needs. In: Int'l Conf. on Comp. Design. (2005) 550–558
4. Seshia, S.A., Li, W., Mitra, S.: Verification-guided soft error resilience. In: Design, Automation and Test in Europe. (2007) 1442–1447
5. Zhao, C., Dey, S.: Improving transient error tolerance of digital VLSI circuits using RObustness COmpiler (ROCO). In: Int'l Symp. on Quality Electronic Design. (2006) 133–140
6. Zhou, Q., Mohanram, K.: Gate sizing to radiation harden combinational logic. IEEE Trans. on CAD **25**(1) (2006) 155–166
7. 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**(3) (2002) 261–271
8. Pellegrini, A., Constantinides, K., Zhang, D., Sudhakar, S., Bertacco, V., Austin, T.: CrashTest: A fast high-fidelity FPGA-based resiliency analysis framework. In: Int'l Conf. on Comp. Design. (2008) 363–370
9. 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
10. 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
11. Fey, G., Drechsler, R.: A basis for formal robustness checking. In: Int'l Symp. on Quality Electronic Design. (2008) 784–789
12. Fey, G., Sülflow, A., Drechsler, R.: Computing Bounds for Fault Tolerance using Formal Techniques. In: Design Automation Conf. (2009)
13. Cook, S.: The complexity of theorem proving procedures. In: 3. ACM Symposium on Theory of Computing. (1971) 151–158
14. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1579 of LNCS., Springer Verlag (1999) 193–207
15. Eén, N., Sörensson, N.: An extensible SAT solver. In: SAT 2003. Volume 2919 of LNCS. (2004) 502–518
16. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Design Automation Conf. (2001) 530–535
17. 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.).
18. Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp. **35**(8) (1986) 677–691