Noname manuscript No.
(will be inserted by the editor)

Feasibility Analysis for MEU Robustness Quantification by Symbolic Model checking

Souheib Baarir · Cécile Braunstein · Emmanuelle Encrenaz · Jean-Michel Ilié · Isabelle Mounier · Denis Poitrenaud · Sana Younes

Received: date / Accepted: date

Abstract We propose and investigate a robustness evaluation procedure for sequential circuits subject to particle strikes inducing bit-flips in memory elements. We define a general fault model, a parametric reparation model and quantitative measures reflecting the robustness capability of the circuit with respect to these fault and reparation models. We provide algorithms to compute these metrics and show how they can be interpreted in order to better understand the robustness capability of several circuits (a simple circuit coming from the VIS distribution, circuits from the itc-99 benchmarks, a CAN-Bus interface).

1 Introduction

Dependability analysis is a major concern of embedded hardware designers, when VLSI circuits are assumed to be submitted to particle strikes, electromagnetic interferences and other signal integrity problems. This could result in unsafe soft errors that may drastically corrupt the expected behaviors, although no damage degrades the hardware. This problem becomes even more critical with current transistor technology shrinking down below 40 nm ; the impact of the perturbation is stronger than with older technologies. With this technology shrinking, a particle impact may modify simultaneously several signals and the cross-talk phenomena, induced by simultaneous signal edges are increased with the reduced distance between signals in recent technologies.

Analyzing the application-level robustness with respect to soft errors is usually carried out by means of fault injection campaigns. Fault injection is based on simulation [20] or hardware emulation [1] of (VHDL) design models in which faults have been injected by different means, producing either ”mutant descriptions” [25], or introducing ”saboteur blocks” [20].

Université P. & M. Curie, LIP6 - CNRS UMR 7606 - 4 Place Jussieu, Paris, France E-mail: first.last@lip6.fr

Emulation-based Approaches. In practice, fault occurrences may cause tons of possible error configurations to be studied, therefore, exhaustive fault injection campaigns are often considered as unaffordable. As a consequence, standard approaches consider restrictive cases, such as a single occurrence of fault, e.g. one bit-flip in the Single Event Upsets approach (SEU) or one erroneous signal edge in the Single Event Transients approach (SET). The development of efficient FPGA-based emulation tools and innovative fault injection [26] inside RAM-FPGA lead significant improvements in the robustness analysis of complex systems: in [17], weak parts of a PIC micro-controller are identified by autonomous fault emulation and hardened. The emulation campaigns ran over 80 million of SEU faults and presented a failure rate lower than 1% with a hardening of 24% of the circuit flip-flop. However, these campaigns represent only a fragment of all possible SEU failures, and computed values lack of confidence margins. *Statistical Fault Injection (SFI)* proposes to randomly select a reduced number of the possible erroneous configurations and, by statistical analysis, gives robustness measures and confidence margins on the results [24] [10].

Analytical VS Behavioral Formal Methods. We try to obtain *guaranteed results* based on the use of formal methods. Formal-based analysis framework establishes rigorously the robustness level of a system, for a given fault model and a particular robustness criterion. Several fault models and repairing capabilities may be integrated into the same formal framework and one can compare several robustness criteria. Among formal robustness analysis approaches, one can distinguish between *analytical* and *behavioral* approaches. The first one considers the probability of a signal to be faulted, and evaluates the probability of the output to be faulted due to the propagation of the faulted input [22] [9]. This method essentially applies to combinatorial circuits subject to single transient fault and is the basis for selective hardening of small circuits [16] up to medium size circuits (with approximations, [27]). Behavioral approaches on the other hand ground the robustness analysis on the behavioral analysis of the system, seen as (potentially diverging) sequences of states. The results obtained are very rich: as it will be presented in this paper, the failure recovery can be defined as a behavioral pattern, the potential or unavoidable reparation can be stated, the repairing speed can be guaranteed; these information provide a deep understanding of the repairing capabilities of the circuit, which can guide the designer to select the most important part to be hardened. Obviously, this latest approach cannot produce results for systems being as big as those analyzed by simulation or emulation, and it is limited to small designs.
Related Works on Behavioral Formal Approaches. In the trend of the initial proposal of [23], some recent papers describe preliminary solutions to apply behavioral formal approaches in the context of dependability evaluation.

The approach of [21] focuses on *measuring the quality of fault-tolerant designs*. The analyzed circuit contains error detection or correction logic and the purpose is to evaluate the efficiency of such logic against injected faults (SEU). Faults are categorized into several classes depending on the severity of their consequences. BDD based symbolic model checking techniques are used to count the fault of different classes. These classes are analyzed to evaluate the coverage of the error detection/correction logic.

In [28], a circuit is considered to be robust if, after a fault injection, the initial specification is still verified. In this context, the robustness evaluation highly depends on the accuracy of the initial specification. Soft errors are considered in latches, using the *SEU* error model. As a consequence, there are as many formal models as faulted circuits such that only one latch is submitted to fault injection. The model checker SMV is then used to check whether the formal specification of the original circuit still holds in each case, thus indicating whether the corresponding latch must be protected or not.

The aim of papers [14, 15] is to give lower and upper bounds measures of robustness circuits subject to SET (or restricted MET) fault model (faults are injected into combinatorial gates and may propagate into latches). The robustness criterion is behavioral equivalence between the faulted and golden model and its evaluation reduces to bounded sequential equivalence checking. The consequence of injected fault applied on a component is analyzed along a bounded sequence. Along this sequence, this fault may have propagated through the primary outputs (the faulted component is not robust) or not. In this last case, the fault may appear later (the faulted component is unclassified) or never (the faulted component is robust). Each faulted component is identified according to this classification (robust, non robust, unclassified). The proportion of these three sets gives lower and upper bounds for the robustness of the circuit.

All of these methods are adapted for the analysis of small systems ranking up to 20 or 50 flip-flops, manageable with BDD or SAT model-checking techniques.

SEU vs. MEU fault models. With the shrinking of transistor technology, a particle impact may modify several signals or register bits simultaneously. Moreover, electromagnetic perturbations affecting spatial applications may cause several disturbances occurring at different instants. We have to consider both the spatial multiplicity and the temporal multiplicity of faults ; current faults models distinguish between the spatial multiplicity: Single event Upset (SEU) considers a unique bit-flip at a unique instant and Multiple Event Upset (MEU) supposes simultaneous bit-flips in several registers [3]. The current fault injection campaigns and formal approaches rarely address the combination of spatial and temporal multiplicity.

In this paper, we aim at considering *Multiple bit-flips*, whatever the fault origin, event upset (EU) or event transient (ET), and in both dimensions, temporal *and* spatial.

Our Contributions. We present three contributions:
• We propose to analyze the circuit's robustness under a general fault model that deals with both the spatial and temporal multiplicity of faults occurrences. We propose two fault models in the same setting:

– A mono-spatial and mono-temporal multiplicity fault occurrence, corresponding to the standard SEU fault model;
– A multi-spatial and multi-temporal multiplicity fault occurrence (initially described in [2]), that is a generalization of the standard MEU (which corresponds to a multi-spatial and mono-temporal multiplicity fault occurrence).

The model we built encompasses all SEU or all generalized MEU in a unique description. With this approach, we do not need to enumerate all particular cases.

• We concentrate on analyzing *the self-healing capabilities* of circuits, under the general fault model we propose. The repairing model we define refers to *self-stabilization* [12, 13]: a system is self-stabilizing if, from any possible configuration reached after a period of particle strikes, once no perturbation occurs anymore and for any evolution of the system, the system progresses towards a desired set of configurations, named *Safe Configurations*, ensuring that subsequent executions are correct. In many applications (video-stream, weak synchronization schemes, ...), these weak robustness models are acceptable. In our proposal, the *Repairing Sequences* and *Safe Configurations* are specified by a repairing automaton that determines the robustness level required by the designer. The repairing automaton offers a more general framework for evaluating robustness than the strict comparison with the non-faulted model or its specification as done in previous cited works ([28, 14, 15]).

• Moreover, we introduce new measures to help designers choosing part of the design that has to be hardened and our method allows for comparison of different implementations of a component with regards to its robustness level. Based on symbolic model-checking and SAT solving, we show how robustness can be quantified in two new interesting measures:

- The number of *potentially or eventually repairable states* yields information about the self-healing capabilities and it may guide the search of minimal subsets of registers to be protected;
- The *healing speed of a robust circuit* allows one to compare several functionally-similar circuits, in order to select the most rapid to self-repair.

The circuit is modeled at the register transfer level (RTL). Synchronous digital circuits are assumed to be described in VHDL or Verilog, however, there is no assumption concerning the functionality of the circuit. We experienced these measures, under our fault models, on several examples: several versions of the gcd computation, described in the VIS distribution [18], some circuits of the itc-99 benchmark suite, and a CAN bus interface. We analyze the robustness of these circuits wrt. these measures and explain how their interpretation can guide the designer to select one version of a design instead of another – less robust – one, or determine a subset of the registers to be protected to ensure a total robustness. The circuits being analyzed are manageable with BDD- or SAT-based model-checking techniques, hence are of comparable size as these given in [21, 28, 14, 15]. In the cited papers, no results are given about MEU faults with temporal multiplicity, nor about reparation speed.

Structure of the paper. The remaining of the paper is structured as follows: section 2 provides basic definitions. Sections 3 and 4 introduce our fault and reparation models respectively. The two measurements and their computation are described in section 5. Several case studies are presented in section 6. Section 7 concludes the paper.

2 Preliminaries

Our approach is applied on circuits described at the RTL level. The representation of the transition and output functions is automatically extracted and easily transformed into the input formats of the verification tools. Hence, a circuit is defined as follows.

Definition 1 (Sequential circuit [6]) A *sequential circuit* C is defined by a tuple $\langle I, O, R, \delta, \lambda, \mathbf{R}_0 \rangle$, where,

- I is a finite set of boolean inputs signals.
- O is a finite set of boolean outputs signals.
- R is a finite set of boolean sequential elements (registers).
- $\delta : 2^I \times 2^R \rightarrow 2^R$ is the transition function.
- $\lambda : 2^R \rightarrow 2^O$ is the output function.
- $\mathbf{R}_0 \subseteq 2^R$ is the set of initial states.

States (or configurations) of the circuit correspond to boolean configurations of all the sequential elements. From now on, let $C = \langle I, O, R, \delta, \lambda, \mathbf{R}_0 \rangle$ be a sequential circuit.

We define a *sequence* of a circuit as an infinite word on the alphabet $2^I \times 2^R \times 2^O$ satisfying the reachability conditions.

Definition 2 (Sequence) Let $\sigma = \langle \mathbf{i}_0, \mathbf{r}_0, \mathbf{o}_0 \rangle, \langle \mathbf{i}_1, \mathbf{r}_1, \mathbf{o}_1 \rangle, \ldots$ an infinite word such that $\forall j \geq 0, \mathbf{i}_j \in 2^I, \mathbf{r}_j \in 2^R$ and $\mathbf{o}_j \in 2^O$. σ is a sequence of C if

1. $\mathbf{r}_0 \in \mathbf{R}_0$ and,
2. $\forall j \geq 0$, $\mathbf{r}_{j+1} = \delta(\mathbf{i}_j, \mathbf{r}_j)$ and $\mathbf{o}_j = \lambda(\mathbf{r}_j)$.

Moreover, a *finite sequence* is defined as usual.

However, not any input sequence may be considered: hypothesis on the environment are modeled as fairness constraints on the inputs. In our context, we limit ourselves to weak fairness constraints as defined in [7]. In the following and when necessary, we will precise how these constraints are taken into account.

3 The fault model

When a circuit is submitted to a peak of particle strikes, *bit-flips* may occur within the sequential elements in the circuit. Keeping in mind that the hardening of all the registers is often not feasible or costs too much, our intention is to select and protect only some of them against bit-flips. We claim that studying different strategies of protection helps designers to identify the set of registers to be hardened. To be exhaustive hence formal, we must consider that bit-flips may occur at any state and within one or more registers.

- Faults are *bit-flips* occurring within the sequential elements of unprotected registers.

– A fault may affect multiple sequential elements at the same time.
– Different faults may occur at different times.

The following definition of MEU error states accords with our **MEU fault model**, enabling all spatial and temporal multiplicities of faults, with respect to the unprotected registers.

Definition 3 (MEU Error states) Let $P \subsetneq R$ be a set of protected registers of a circuit C. The set $Error_{meu}(P)$, is built inductively as the smallest subset of 2^R satisfying the following assertions :

1. $\mathbf{R}_0 \subseteq Error_{meu}(P)$
2. $\mathbf{r} \in Error_{meu}(P) \Rightarrow \{\mathbf{r}' \in 2^R \mid \forall p \in P, \mathbf{r}'[p] = \mathbf{r}[p]\} \subseteq Error_{meu}(P)$
3. $\mathbf{r} \in Error_{meu}(P) \Rightarrow \{\mathbf{r}' \in 2^R \mid \exists \mathbf{i} \in 2^I \text{ such that } \mathbf{r}' = \delta(\mathbf{i}, \mathbf{r})\} \subseteq Error_{meu}(P)$

The first item of Definition 3 forces to consider the reachable states as erroneous. Actually, it could appear that a bit-flip may cancel the effect of another bit-flip, therefore all the states of the circuit must be suspected. In the second item, error states stem from bit-flips in an unprotected registers. Lastly, bit-flip propagations are taken into account, towards all of registers accordingly to the transition relation δ. As a consequence, the size of $Error_{meu}(P)$ encompasses the number of states directly altered by some bit-flips.

Computing the *Error states* is the starting point of our measurements. It is easy to prove that each of the error states derives from a reachable state of the circuit, from which a series of sequences are applied, each one followed by a fault injection occurrence. Our implementation is based on an adaptation of the symbolic reachability procedure designed for symbolic model checkers and starts from the reachable set of states as the first set of error states. From such a set, an effective injection procedure is called to generate all the newly possible error states, by means of a symbolic relaxation of the binary values within the unprotected registers. The reachability procedure is used to implement the error propagation through the registers, therefore adding new error states to the previously visited ones implies to iterate both the reachability and injection procedures, until no new error state can be computed.

Compare to the fault injection approaches of [21,28,15], our method does not require to use any additional component nor source code modification. Instead, the computation of the reachable states is augmented to consider the extra states and transitions induced by faults. For sake of memory and time computation, our approach can easily be adapted to compute more restricted set of error states. This is demonstrated in some of the measurements presented in Section 6, where each error state is due to a unique fault occurrence, hence implementing the **SEU fault model**. This causes a drastic reduction of the number of error states, and injection procedure is only applied once from the standard Reachability set.

Definition 4 (SEU error states) Let $P \subsetneq R$ be a set of protected registers of a circuit C. The set of reachable states under unique bit-flips is named $Error_{seu}(P)$.

$$Error_{seu}(P) = \{\mathbf{r} \in 2^R \mid \exists \mathbf{r}' \in reach, \exists f \in (R \setminus P) \text{ s.t. } \mathbf{r}[f] \neq \mathbf{r}'[f] \wedge \forall r \in R \setminus \{f\}, \mathbf{r}[r] = \mathbf{r}'[r]\}$$

where *reach* represents the set of reachable states of the circuit, in case of no fault occurrences.

4 Repairing model

Let us now study the ability of the circuit to recover, directly after the bit flip period and assuming that no more fault can occur during this stage. The circuit is then in one of the error states. From such a state, the circuit designer could be interested to know whether the circuit can reach a safe state from which the circuit certainly recovers.

However, the designer which knows more upon the circuit and its application context, would agree the possibility to introduce some additional constraints that do not only concern the reach *safe* states but also the way to reach them. For instance, while progressing, the circuit should be expected to visit some *mandatory* states and in contrast, to avoid some *critical* ones. Such a specification can be described by using the theory of finite state automata [19] : a finite sequence starting from an error state is considered as repairing if it is recognized by a finite automaton, namely, a *repairing automaton* whose accepting states are safe states. Repairing automaton and its associated notion of *repairing sequence* are defined as follows.

Definition 5 (Repairing Automaton) A *repairing automaton* for C is defined by $A = \langle S, T, s_0, F \rangle$ where :

- S a finite set of states.
- $T \subseteq S \times 2^R \times S$ a finite set of labeled transitions.
- s_0 an initial state.
- F a finite set of accepting states.

Definition 6 (Repairing Sequence) Let $\sigma = \langle \mathbf{i}_0, \mathbf{r}_0, \mathbf{o}_0 \rangle, \langle \mathbf{i}_1, \mathbf{r}_1, \mathbf{o}_1 \rangle, \ldots \langle \mathbf{i}_k, \mathbf{r}_k, \mathbf{o}_k \rangle$ be a finite sequence of C such that $\mathbf{r}_0 \in Error(P)$. σ is said to be *repairing* if it is recognized by the repairing automaton $A = \langle S, T, s_0, F \rangle$: $\exists \langle s_0, s_1, \ldots, s_{k+1} \rangle \in S^{k+2}$ s.t. $\forall 0 \leq j \leq k, \langle s_j, r_j, s_{j+1} \rangle \in T$ and $s_0 \in S_0$ and $s_{k+1} \in F$. Moreover, the sequence σ is said to be *elementary repairing* if no one of its strict prefixes is repairing.

The designer is intended to specify the repairing sequences like an automaton. The task consists in labelling the transitions of an automaton by states of the circuit, in order to describe the sequences that are considered as repairing ones. To have a concise representation of the automaton, we labelled the transitions by sets of states. In Figure 1, *Safe*, *Required* and *Forbidden* respectively stand for the sets of safe, mandatory and critical states. An over-lined set represents the relative complement

of the set with respect to 2^R. Therefore the automaton of Figure 1 recognizes the sequences leading to a state that belongs to *Safe* visiting along the way at least one state belonging to *Required* but no state belonging to *Forbidden*.

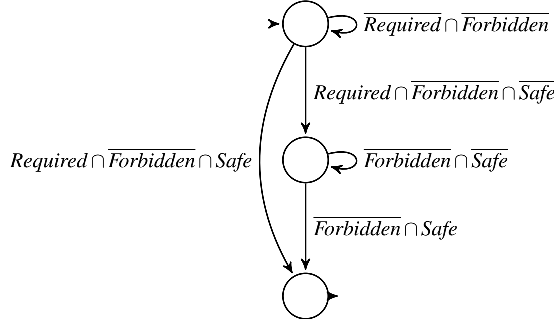

Fig. 1 A repairing automaton.

When the only constraint is that the circuit recovers in a state reachable when no fault occurs, no set *Required* and *Forbidden* is specified and *Safe* equals to *reach*. The associated automaton is given by Figure 3 in section 6.

Figure 2 highlights in a 3 levels tree, the partial execution of some circuit, starting from an error state. The membership of states to sets $Error(P)$, *Safe*, *Required* and *Forbidden* is represented with different shapes. The sequences σ^a, σ^b, σ^f, σ^j and σ^k are not repairing. Actually, both σ^a and σ^b visit a state that belongs to *Forbidden*, σ^f does not visit a state that belongs to *Required* and both σ^j and σ^k do not reach a state that belongs to *Safe*. The other sequences are effective repairing sequences, but only σ^c and σ^i are said to be elementar because σ^d, σ^e, σ^g and σ^h contain a repairing sequence of depth 2.

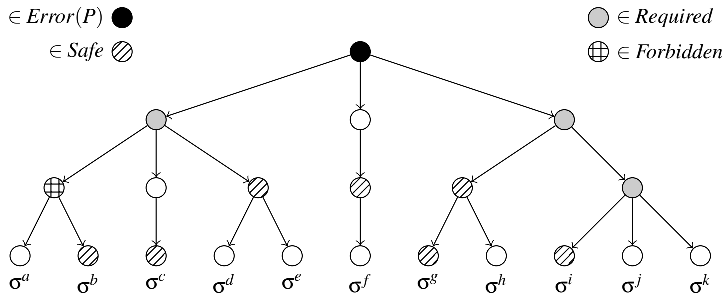

Fig. 2 A part of the execution tree starting from an error state.

We now refine the notion of repairing sequences, by distinguishing potentially and eventually reparable states. This requires an analysis of the possible sequences that can be generated from an error state.

Definition 7 (Potentially and Eventually Reparable States)
An error state is said to be potentially reparable if it is the initial state of at least one repairing sequence. It is said to be *eventually reparable* if every infinite sequence that can be generated from it is prefixed by a repairing sequence.

The finite sequences are always considered as prefixes of some infinite sequences, moreover, the infinite sequences are built under some fairness constraint (optionally empty) expressed on the inputs of the circuit. The error state of Figure 2 is potentially but not eventually reparable.

5 Robustness Quantification of a Circuit

This section presents two metrics to quantify the robustness of a circuit according to our fault and repairing models. We also provide their algorithmic computation.

5.1 State-based quantification

The first metric consists in the ratio of the number of reparable states out of the number of error states. This measure is determined considering potentially or eventually reparable states. To perform this measure, we have to compute the set $Error(P)$ and to execute the circuit from these states with respect to the repairing automaton. The executions that lead to an accepting state of the automaton are the repairing ones. We reduce the computation of the ratios to the counting of some reachability sets using a fair CTL model checker. To fulfil this counting, we build an instrumented circuit whose reachable states corresponding to repairing states are easily identifiable.

Let $A = \langle S, T, s_0, F \rangle$ be a deterministic and complete repairing automaton. We consider $AC = \langle I_{AC}, O_{AC}, R_{AC}, \delta_{AC}, \lambda_{AC}, \mathbf{R}_{AC_0} \rangle$ a sequential circuit encoding A. We impose that the output set is a singleton, $O_{AC} = \{o_{ac}\}$, whose unique signal is set to one if and only if the encoded state belongs to F. By construction, the transitions in A depend on the current state of C.

We introduce two notations. Let R_1 and R_2 be two sets of registers, $\mathbf{a} \in 2^{R_1}$ and $\mathbf{b} \in 2^{R_2}$ be two configurations. We denote by $\mathbf{a}.\mathbf{b}$ the concatenation of the two configurations. Let be $a \in R_1$, we denote by $\mathbf{a}[a]$ the projection of the configuration on its component a.

Definition 8 (Instrumented circuit) Let $C = \langle I_C, O_C, R_C, \delta_C, \lambda_C, \mathbf{R}_{C_0} \rangle$ be a circuit, $P \subsetneq R$ a set of protected register and $AC = \langle I_{AC}, O_{AC}, R_{AC}, \delta_{AC}, \lambda_{AC}, \mathbf{R}_{AC_0} \rangle$ a sequential circuit encoding the repairing automaton of C. In consequence, we set $I_{AC} = R_C$ and $O_{AC} = \{o_{ac}\}$. The instrumented circuit $C \otimes AC = \langle I, O, R, \delta, \lambda, \mathbf{R}_0 \rangle$ is defined by:

- $I = I_C$, $O = O_C \cup O_{AC}$ and $R = R_C \cup R_{AC}$
- $\forall \mathbf{i} \in 2^{I_C}, \forall \mathbf{r}_c \in 2^{R_C}, \mathbf{r}_{ac} \in 2^{R_{AC}}$
  - $\forall r_c \in R_C, \delta(\mathbf{i}, \mathbf{r}_c.\mathbf{r}_{ac})[r_c] = \delta_C(\mathbf{i}, \mathbf{r}_c)[r_c]$
  - $\forall r_{ac} \in R_{AC}, \delta(\mathbf{i}, \mathbf{r}_c.\mathbf{r}_{ac})[r_{ac}] = \delta_{AC}(\mathbf{r}_c, \mathbf{r}_{ac})[r_{ac}]$
- $\forall \mathbf{r}_c \in 2^{R_C}, \mathbf{r}_{ac} \in 2^{R_{AC}}$

  - $\forall o_c \in O_C, \lambda(\mathbf{r}_c.\mathbf{r}_{ac})[o_c] = \lambda_C(\mathbf{r}_c)[o_c]$
  - $\lambda(\mathbf{r}_c.\mathbf{r}_{ac})[o_{ac}] = \lambda_{AC}(\mathbf{r}_{ac})[o_{ac}]$
- $\mathbf{R}_0 = Error(P) \times \mathbf{R}_{AC_0}$

Thanks to the output signal o_{ac}, we are able to detect if the instrumented circuit reaches a repaired configuration. According to the repairing automaton, the set of repaired configurations is defined by $Repaired = \{\mathbf{r} \in 2^R \mid \lambda(\mathbf{r})[o_{ac}] = 1\}$.

In this context, the potentially repairing states verify the CTL formula $\text{EF}_{fair}\,Repaired$ and the eventually ones verify $\text{AF}_{fair}\,Repaired$ (where *fair* represents the fairness constraint). In the following, a CTL formula denotes the set of configurations of the instrumented circuit satisfying it. We denote ν_{pot} (resp. ν_{ev}) the ratio of potentially (resp. eventually) reparable states out of the number of error states. The two ratios are defined as follows.

$$\nu_{pot} = \frac{\left|\text{EF}_{fair}\,Repaired \cap \mathbf{R}_0\right|}{|\mathbf{R}_0|} \qquad \nu_{ev} = \frac{\left|\text{AF}_{fair}\,Repaired \cap \mathbf{R}_0\right|}{|\mathbf{R}_0|}$$

The computation of ν_{pot} and ν_{ev} is performed using a CTL model-checker on the instrumented circuit.

5.2 Sequences-based quantification

The idea of the second metric is to highlight the velocity of the circuit to recover from an error state. This last metric can be used to compare different architectures with the same functionalities.

We focus on the elementary repairing sequences containing no loop. We want to find out the minimal and maximal length of such sequences. We limit ourselves to sequences without loop since, if an elementary sequence contains such a loop, the maximal bound does not exist. Moreover, for sequences without loop, we aim at counting the number of elementary repairing ones for each possible length between these two bounds.

The computation of these bounds is performed by solving SAT problems iteratively [4,29]. The counting of elementary repairing sequences of a given length is then reduced to a #SAT problem [5].

We now present the SAT encoding of sequence without loop as well as elementary repairing ones. In the following, $\mathbf{r}_i$ denotes a vector of propositional variables representing the state configuration of the instrumented circuit at step i. We use the instrumented circuit of Definition8.

Sequences without loop of the instrumented circuit C of length k satisfy the following SAT problem, denoted $WithoutLoop(k)$:

$$WithoutLoop(k) = [\mathbf{r}_0 \in \mathbf{R}_0] \wedge \left[\bigwedge_{0 \leq j < k} (\exists \mathbf{i} \in 2^I, \mathbf{r}_{j+1} = \delta(\mathbf{i}, \mathbf{r}_j))\right] \wedge \left[\bigwedge_{0 \leq j < l \leq k} \mathbf{r}_j \neq \mathbf{r}_l\right]$$

Moreover, elementary repairing and sequences are solutions of the following SAT problem, denoted $ElementaryRep(k)$:

$$ElementaryRep(k) = [WithoutLoop(k)] \wedge [\mathbf{r}_k \in Repaired] \wedge \left[\bigwedge_{0 \leq j < k} \mathbf{r}_j \notin Repaired \right]$$

Algorithm 1 computes the number of elementary repairing sequences without loop, executable on the instrumented circuit up to the diameter of its reachability graph.

Algorithm 1: Computation of the Repairing Sequence Repartition

Input: C: an instrumented circuit;
Output: t: array of Integer;
$k = 0$;
while $SAT(WithoutLoop(k))$ **do**
  $t[k] = \#SAT(ElementaryRep(k))$;
  $k = k + 1$;
return(t);

The expensive operation in Algorithm 1 is the computatfion of #SAT whose number of calls is related to the diameter of the reachability graph. One way to downscale the complexity is to replace #SAT by SAT; the obtained procedure can be used to determine the minimal (l_{min}) and maximal (l_{max}) bounds of repairing sequences.

6 Case studies

As illustration, we provide a set of experiments enlightening the different measures we propose and how they can be interpreted in terms of robustness. For the computation of ν_{pot}, ν_{ev}, l_{min} and l_{max} we have adapted the VIS model-checker [18]. Also, we integrated to VIS the sharpSAT solver [31] in order to count elementary repairing sequences.

The measures are performed on several versions of the `gcd` example from the VIS distribution, a CAN-bus [30, 11] and part of the ITC'99 benchmarks [8]. For all these case studies, the fault model is the one defined in section 3. The repairing model is based on the repairing automaton of Figure 3 where the set *reach* corresponds to the set of reachable configurations of the circuit without fault and starting from its initial states. This is a typical recovery model conforming to self-stabilizing systems. For this kind of circuit, from any reachable configuration and when no fault occur anymore, if a new computation is asked, then it is guaranteed that it will finally produce the correct result.

The common columns of the Tables 1, 3, 4 and 5, are C that denotes the circuit, $|reach|$ the number of reachable states from the initial states when no fault occurs, the two considered fault models denoted M for "multiple faults" and U for "single fault", $|Error(P)|$ the number of error states according to the fault model, ν_{pot} and ν_{ev} the potentially and eventually repairing ratios and l_{min} and l_{max} the minimal and maximal bounds of elementary repairing sequences.

For all these experiments, the times spent to compute the presented ratios are negligible quantities, thus are not figured. Also, the times to compute the lengths of

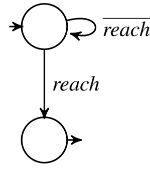

Fig. 3 The repairing automaton for cases study.

sequences are not represented. In fact, the waiting time for circuits having $l_{max} < 100$ is very weak, but we stop computations for circuits having bigger values since the time spent can be huge.

6.1 `gcd`

The measures are performed on several versions of the `8-bits gcd` example from the VIS distribution. This circuit computes the greatest common divisor of two operands (inputs *a* and *b*) on request (input signal *start*). The computation takes several cycles (output signal *busy* is set to 1) and when the result is computed, it is produced on output *o* (*busy* is set to 0). The computation makes use of three internal registers (*lsb*, *x* and *y*). The implemented algorithm is presented in Algo 2. For all experiments, no register is protected, $P=\emptyset$.

In its original version and when no fairness constraint is considered (`gcd` in Table 1), the model satisfies several properties. In particular, as soon as there is no more fault and whatever bit-flips had occurred during the fault injection period, and even if the current produced result is erroneous, the computation step always terminates. Moreover, when a new computation is asked, its result will be correct. However, if no new computation is asked, the circuit may stay in its erroneous state set: 79% of erroneous configurations of `gcd` do not return systematically into a correct behavior.

On the contrary, when the circuit behaves under a fairness constraint (`gcd`$_{fair}$), imposing that a new computation will always be asked in the future (GF(*start*)), 100% of error states are eventually repairable even if no register is protected during the fault strike period.

The computed ratios are the same whatever the fault model, they depend on the structure of the circuit and on the fairness constraint (if we consider it or not).

Let us now consider a modification of the circuit that invalidates this result (see row `gcd-v1`$_{fair}$). This modification is obtained considering the grey part of Algo. 2. When no fault occurs, this modification has no incidence on the behavior of the circuit. Indeed, when $lsb = 14$ then, by successive divisions and substractions, the condition $x = y \vee x = 0 \vee y = 0$ is reached and the computation terminates. On the contrary, when faults occurred, some 1's may have been introduced into *x* or *y*, and bit-flips in *lsb* may have induced some bits in *x* and *y* not to be considered and the condition $x = y$ may never be reached. In the original algorithm, the increment of *lsb* may reset it to 0 and this insures that condition bounding the computation will be reached after a last examination of *x* and *y*. The modification of `gcd-v1`$_{fair}$ prevents this last examination and, as a consequence, the computation may not terminate. If we consider the

C	Fault model	$\lvert reach \rvert$	$\lvert Error(P) \rvert$	ν_{pot}	ν_{ev}	l_{min}	l_{max}
gcd	M	$1.37e^{05}$	$2.1e^{06}$	100%	21%	0	15
	U		$6.8e^{05}$				
gcd_{fair}	M		$2.1e^{06}$	100%	100%	0	15
	U		$6.8e^{05}$				
$gcd-v1_{fair}$	M		$2.1e^{06}$	98%	98%	0	15
	U		$6.8e^{05}$	99.7%	99.7%		
$gcd-v2_{fair}$	M	$3.04e^{05}$	$5.36e^{08}$	100%	100%	0	17
	U		$3.8e^{06}$				

Table 1 Robustness Measures of several versions of `gcd`.

multi faults model, 2% of error states never reach a repairing state while considering the single fault model they are $0,03\%$. With a multi faults model and for a given x, we have more possible error states with $x \neq y$ than with a single fault model. Such error states are all not eventually repairable. Therefore the ratio ν_{ev} is greater with a multi faults model than with a single fault one.

A way to prevent the infinite loop introduced in version `gcd-v1`$_{fair}$ consists in protecting all registers (x, y and lsb). An alternative consists in forcing the condition $(busy \wedge x \neq y \wedge x \neq 0 \wedge y \neq 0)$ to become false after at most n consecutive cycles (where n represents the maximal number of steps needed for a correct computation, here $n = 15$). The results of this version (named `gcd-v2`$_{fair}$) show that 100% of error states are eventually reparable even if no register is protected. These results are valid with the multi faults model as well as with the mono fault one.

Actually, the original `gcd` algorithm needs at most $n = 15$ clock cycles to satisfy the condition $T = (x = y \vee x = 0 \vee y = 0)$. Hence, the version `gcd` will reach a state of *reach* in at most 15 clock cycles whatever the starting state of the design: if we note (x,y,lsb) a state of the gcd, then by construction we can easily check that $\{(x,y,lsb) \mid x = y \vee x = 0 \vee y = 0\} \subseteq reach$. For the version `gcd-v2`$_{fair}$, the situation is different: after a bit-flip, and because the modification of the constraint T, becoming $T_1 = (T \vee n > 15)$, the computation will (in any case) end after at most 15 cycles, but at this point T may still not be satisfied. Hence, the obtained state may not to be in *reach*. In this last case, the circuit needs one more cycle to set *busy* to 0 and another one to re-initialize the registers; this forces the circuit to comeback to a state of *reach*.

C	Cycles				
	0	1	2	3	4
`gcd`	$4{,}71e^{-15}$	$7{,}86e^{-12}$	$4{,}92e^{-10}$	$1{,}47e^{-7}$	$9{,}85e^{-5}$
`gcd-v2`	$5.23e^{-21}$	$2{,}21e^{-17}$	$3{,}91e^{-15}$	$8{,}70e^{-13}$	$4{,}28e^{-10}$

C	Cycles			
	5	6	7	8
`gcd`	0,05	0,94	-	-
`gcd-v2`	$1{,}54e^{-7}$	$1{,}22e^{-5}$	0,002	0,99

Table 2 Distribution of the elementary repairing sequences for `gcd`.

To compare the velocity of repairing between several versions of the gcd, we need to apply our sequences-based quantification approach. Table 2 highlights the obtained results: the used models are $\texttt{gcd}_{fair}$ and $\texttt{gcd-v2}_{fair}$ of Table 1, but reduced to 4-bits designs under the multi faults model. For the $\texttt{gcd}_{fair}$ model, l_{min}=0 and l_{max}=6 while for the $\texttt{gcd-v2}_{fair}$ l_{min}=0 and l_{max}=8 (the difference between the two values of l_{max} is explained as for the 8-bits designs model).

For each model we compute the ratio of the number of elementary repairing sequences at cycle i ($|S(i)|$) to the sum of elementary repairing sequences ($\sum_{i=l_{min}}^{l_{max}} |S(i)|$). We observe that, for the two models, almost all elementary repairing sequences are concentrated on the last cycle.

This simple case study illustrates the soundness of our metrics for the quantification of the robustness of circuits: by means of state-based approach, we can establish whether the circuit is robust ($\nu_{ev} = 100\%$) or not. Here, circuits gcd and $\texttt{gcd-v2}_{fair}$ are robust, yet $\texttt{gcd-v1}_{fair}$ is not. Furthermore, the sequences-based approach allows us to distinguish between robust circuits, presenting the same functionalities. Here circuit gcd recovers more quickly than $\texttt{gcd-v2}_{fair}$, hence the former offers a better robustness velocity criterion.

Algorithm 2: Original algorithm of gcd and, in gray, modification v1

```
Input: start : signal; a,b : integer[8];
Output: busy : signal; o : integer[8];
Registers: lsb : integer[3]; x,y : integer[8];
begin initialisation
|  x ← 0; y ← 0; lsb ← 0; busy ← 0; o ← 0;
end initialisation
for each positive edge clock do
    if (start ∧ ¬busy) then
        x ← a; y ← b; lsb ← 0; busy ← 1;
    else if (busy ∧ x ≠ y ∧ x ≠ 0 ∧ y ≠ 0) then
        tmp : integer[2];
        tmp ← 2.x[lsb] + y[lsb];
        switch tmp do
            case b'00
                if (lsb < 15) then
                    lsb ← lsb + 1;
            case b'01
                x ← x/2;
            case b'10
                y ← y/2;
            case b'11
                if (x < y) then
                    y ← (y − x)/2;
                else
                    x ← (x − y)/2;
    else if (busy ∧ (x = y ∨ x = 0 ∨ y = 0)) then
        o ← (x < y ? x : y); busy ← 0;
end for
```

6.2 itc99

The ITC benchmarks are proposed in the context of Automatic Test Pattern Generation (ATPG) [8] [1]. We focus on a coherent subset, called the Torino Benchmarks, that were proposed to ease evaluations and comparisons of research tools. It consists of twenty-two various circuits, all described at both RTL and gate levels. We evaluate the robustness for thirteen of these circuits, found as the first ones in the bench suite and together included into the VIS distribution. Among them, we report on the robustness evaluation of circuits b03, b08, b09, b10, b11 and b13. Circuits b01, b02, b06 and b07 have been evaluated but their interest is limited since they present a very small state space. Circuits b04 and b12 cannot be evaluated with VIS since the computation of the reachable state space blows up. As the obtention of the state space is mandatory to compute the set of error states, no robustness measure is obtained.

Almost all the tested circuits appear to be non robust. Let us recall that totally robustness property requires 100% ν_{ev} rate for the MEU fault model. In fact, this result is not really surprising since (one decade ago) the designers were more interested in the satisfaction of soundness properties than in robustness consideration.

Table 3 relates by the measures over 6 circuits, the various situations we obtained. Two new columns are introduced to precise the numbers of combinatorial gates in some circuit (#gates) and the global number of bits in registers ($\#FF$ stands for flip-flop), respectively. In the proposed experiments, the set of protected registers is empty and no fairness constraint is applied to the environment.

C	#gates	#FF	F.M.	$\|reach\|$	$\|Error(P)\|$	ν_{pot}	ν_{ev}	l_{min}	l_{max}
b03	150	30	*M*	2058	$1.1e^{09}$	75%	0.09%	0	>100
			U		42093	95%	49.5%		
b08	168	21	*M*	29186	2097152	100%	14.3%	0	34
			U		241960		29.7		
b09	131	28	*M*	262401	$2.7e^{08}$	100%	97.4%	0	26
			U		$3.1e^{06}$		96.7%		
b10	172	17	*M*	4464	$1.04e^{06}$	19.5%	0.83%	1	>100
			U		45127	60.8%	11%		
b11	366	30	*M*	169630	$2.1e^{09}$	56.2%	3.4%	1	>100
			U		$3.8e^{06}$	94.6%	34.2%		
b13	309	53	*M*	$5.2e^{07}$	$9e^{15}$	48.6%	0.33%	1	>100
			U		$1.5e^{09}$	94.5%	49.6%		

Table 3 Robustness Measures for itc99 suite

About b03, b10, b11 and b13, several measures lead us to think that these circuits do not present a good ability to self-reparation:
- there is an important blow up in the number of states, when considering erroneous states w.r.t. normal reachable ones.
- for both single and multiple faults, all of these circuits can reach error states from

[1] `http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html`

C	F.M.	P	$\|Error(P)\|$	ν_{pot}	ν_{ev}	l_{min}	l_{max}
b03	M	all $\setminus\{ru\}$	3888	100%	100%	0	1
	U		2424				
	M	all $\setminus\{GRANT\}$	10912	100%	100%	0	2
	U		5640				
	M	all $\setminus\{GRANT, ru\}$	40192	100%	100%	0	2
	U		6208				
	M	all $\setminus\{GRANT, coda*\}$	$8.9e^{07}$	100%	0.18%	0	> 100
	U		30336		37.7%		
	M	all $\setminus\{coda*\}$	$2.5e^{07}$	100%	0.18%	0	> 100
	U		24696		23.5%		
	M	all $\setminus\{fu*\}$	$1.22e^{06}$	100%	1.84%	0	> 100
	U		4776		99.5%		

Table 4 Robustness Measures for b03 (detailed)

which no reparation is possible: $\nu_{pot} < 100\%$.
- we were not able to find the maximal length of the elementary reparation sequences, because we stop algorithm 1 beyond 100 length units.
Undoubtedly, the circuit is forced to livelock over numerous unsafe configurations, whichever the inputs.

The circuit b08 seems to be more robust since, even for multiple faults, all of its error states may lead to a reparation and we were able to find the maximal length of elementary repairing sequences. Anyway, only a few proportion of them are totally repairing.

The profile of the circuit b09 is slightly different since the total reparation ratio are close to 100%, for both single and multiple faults. Again, the maximal repairing length has been successfully computed. This circuit has an intrinsic good reparation skill.

Let us show now how the measures we provide can help to identify a subset of registers that may be protected, to ensure a total robustness. Among the former circuits, we select b03, which presents the lowest ν_{ev} ratio. The treated circuit corresponds to a resource arbiter, composed of 30 bits of registers, namely *stato (2), coda (12), GRANT (4), GRANT_O (4), ru (4), fu (4)*. The column *P* precises which of these registers have been protected.

The three first lines of the table show that bit-flippings over the registers *ru* or *GRANT* have no impact on the total robustness of the arbiter: for the SEU and MEU error models, ratio ν_{ev} reaches 100%. In all other tested configurations (see the subsequent lines in the table), a 100% potential reparation is achieved, but not an eventually reparation. We can conclude that *ru* and *GRANT* do not have to be protected.

A design achieving 100% reparation for MEU has also a 100% ratio for SEU. In all other cases, the ratios depend on the number of error states, hence nothing can be deduced.

6.3 CAN Bus Interface

The CAN-Bus protocol (CAN for Controller Area Network) is an ISO standard allowing asynchronous message passing [30]. Originally designed for automotive applications, it is recently used in industrial automation and medical equipment. The protocol transmits serial frames between several emitters and receivers; In order to solve conflicts between them, the protocol makes use of a Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) mechanism . In this section we analyze a parallel to serial and serial to parallel interface of this bus. The system we consider is composed of two parts: an emitter, accepting data on its parallel input port, and outputting a serial CAN-frame; and a receiver, accepting a serial CAN-frame and outputting its data content on its parallel output port.

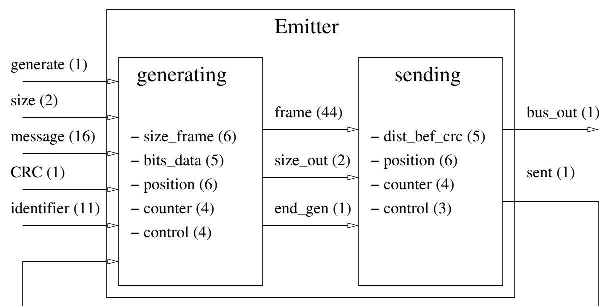

Fig. 4 CAN Bus emitter.

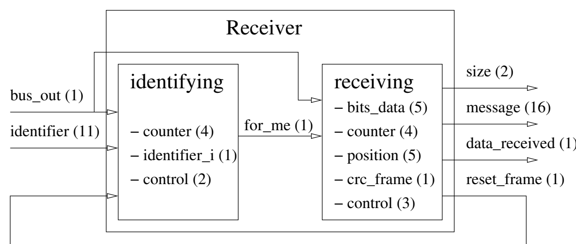

Fig. 5 CAN Bus receiver.

As it is highlighted in Figures 4 and 5, the emitter and the receiver are designed modularly. The emitter is composed of two internal components: a *generating* component that carries on the generation of frames for the CAN-bus; and a *sending* component that serially transmits the generated frame to receivers. A receiver is composed of two internal components: an *identifying* component that determines whether the received frame is addressed to the current receiver station, and a *receiving* component that permits destination stations to store and use the transmitted frame.

SOF	Identifier	RTR	IDE	R0	DLC	DATA	CRC	CRC delimiter	ACK	ACK delimiter	EOF
1–bit	11–bits	1–bit	1–bit	1–bit	2–bits	16–bits	1–bit	1–bit	1–bit	1–bit	7–bits

Fig. 6 Simplified *standard* data frame of CAN Bus

The CAN-bus protocol allows two data frame formats: *standard* and *extended*. The difference comes from the size of the identifier field determining the identity of the receiver, 11-bits for *standard* format and 29-bits for the *extended* format. In this paper, we consider the *standard* one, which is composed of a succession of fields: 1-bit for SOF that indicates the start of frame, 11-bits for the receiver identifier, 1-bit for RTR (Remote Transmission Request) to differentiate between data and control frame, 1-bit for IDE (Identifier Extension bit) to differentiate between *standard* and *extended* frame format, 1-bit reserved, 4-bits for DLC (Data Length Code) to dictate the length by bytes of Data field, 0 up to 32-bits (structured in bytes) for Data, depending on the value of the field DLC, 15-bits for CRC (Cyclic Redundancy Check) used by the receiver to check for errors, 1-bit to delimit the CRC field, 1-bit for ACK, 1-bit as the ACK delimiter, finally 7-bits for EOF specifying the end of the frame. In our study, we simplify some fields in order to reduce the complexity of the verification: the DATA field length is 16-bits instead of 32, and consequently DLC takes 2-bits instead of 4-bits, also CRC is reduced to 1-bit instead of 15-bits. Moreover, we do not consider the bit stuffing mechanism, that were proposed to signal error occurrences over the transmission line (see [11] for a detailed description of the protocol). Hence, the frame format we consider contains 44-bits illustrated in Figure 6.

Our aim is analyzing the robustness of the CAN-Bus protocol, therefore we describe in Verilog a so-called CAN architecture gluing the components of the CAN-Bus together with the components of an emitter and a receiver stations. It appears that VIS is unable to generate the whole state space of this architecture, due to the huge size of the corresponding BDD (The number of nodes exceeds $6e^7$ even when applying dynamic reordering strategies).

We have succeeded in doing the robustness analysis modularly, according the following stages : first, analyze each component separately and try to make them robust by using partial protections on registers ; then, consider the whole CAN architecture and scrutinize the emitter and receiver by applying the faulty and reparation models described in sections 3 and 4. In any case, we add an "environment" fairness constraint on the input signals of each considered component, in order to force the progression of the component. For instance, the sending component is forced to be such that "infinitely often, a new frame is built and sent" .

Table 5, where column P specifies the set of protected registers, presents the obtained robustness ratios. It is worth noting that all computation times are less then 1s, but the one of the Emitter circuit, where it takes an average of half an hour for each result. Looking at these results, we get the following information:

1. Separate analysis of each component (lines *gen*, *send*, *recv*, *ident*):
   Both components *sending* and *identifying* are 100 % robust for any fault model considered. Concerning the components *generating and receiving*, a 100% robustness ratio can be obtained but only if the register "control" is protected. A

C	Fault model	P	$\|reach\|$	$\|Error(P)\|$	ν_{pot}	ν_{ev}
gen	M	control	$1.89e^{11}$	$6.61e^{22}$	100 %	100 %
		-		$7.08e^{22}$	93.33%	93.33%
	U	control		$7.47e^{12}$	100 %	100 %
		-		$8.17e^{12}$	99.05%	99.05%
send	M	control	634	$8.38e^{06}$	100%	100 %
		-				
	U	control		$9.94e^{03}$		
		-		$1.16e^{04}$		
Emitter = gen + send	M	control	$1.27e^{12}$	$1.43e^{28}$	100 %	100 %
	U			$6.99e^{13}$		
recv	M	control	$8.4e^{06}$	$9.62e^{11}$	100 %	100 %
		-		$1.09e^{12}$	87.5%	87.5%
	U	control		$1.39e^{08}$	100 %	100 %
		-		$1.59e^{08}$	96.69%	96.69%
ident	M	control	141	512	100 %	100 %
		-				
	U	control		343		
		-		385		
Receiver = ident + recv	M	control	$2.01e^{08}$	$1.93e^{14}$	100 %	100 %
	U			$3.61e^{09}$		

Table 5 Robustness Measures of different components of CAN Bus

final observation is that all the ratios ν_{pot} and ν_{ev} are very fast to compute, even for large error sets.

2. Analysis of the emitter and receiver stations (lines *Emitter* and *Receiver*).
   Both stations are 100% robust under their environment fairness constraints, when combining their two 100% robust sub-components. This is a particularly interesting result since, in general, the combination of two 100% robust components may not systematically result in a 100% robust assembly. Actually from any two 100% robust components, e.g. $C1$ and $C2$, and their respective reachable state sets, $reach(C1)$ and $reach(C2)$, a robust composed circuit $C1 \times C2$ taken after a perturbation period, must reach one state of its reachable states $reach(C1 \times C2)$. The problem comes from the fact that this last set is included in $reach(C1) \times reach(C2)$, but not necessarily equal to. Thanks to our experiments, we demonstrate that these stations are quite robust.
3. Analysis of the whole CAN architecture :
   Assuming the environment fairness constraints, we check that the following two properties hold: (a) from all the states in $reach(Emitter)$, a new frame will eventually be sent by the Emitter; (b) from all the states of $reach(Receiver)$, a new frame will eventually be received by the Receiver. Combined with item 2, we conclude that the emitter and receiver stations will eventually recover.

The presented experiment showed the ability of a CAN architecture to recover by itself after a perturbation period over the register elements. We exhibited the minimal

set of registers of the elementary components to be protected. The ability to recover after a bounded perturbation period, relaxes the robustness criterion standardly provided by the CRC and bit-stuffing mechanisms, which detect and correct erroneous frames on-the-fly. It is worth noting that our fault model is larger than considering bit-flipping during the serial transmission. The bit-flips also concern the components that generate the frame and the ones that analyze it, before and after the serial transmission. For instance, bit-flips may cause the production of inconsistent frames, and we then prove that after the recovery period, consistent frames are transmitted again.

7 Discussion and conclusion

We propose a novel framework to analyze the robustness of circuits submitted to particles strikes. Our approach allows to cope with *multiple transient faults* captured as bit-flips in sequential elements, and relaxes the robustness criterion. In this paper, we illustrate our methodology applying it to different circuits, showing the soundness of our approach and the interest of our measures.

To the best of our knowledge, MEU is rarely considered when evaluating robustness, since this requires to cope with a combinatorial explosion of fault instants. In this paper, we consider two models of faults, one corresponding to standard SEU, and another one extending MEU to multiple occurrences in time. We presented an original approach to compute efficiently the set of error states for these two models, exploiting the foundations of BDDs symbolic representation and temporal model checking algorithms.

The model of robustness focuses on the *self-healing reparability* of circuits. Among the two metrics we propose, the first one quantifies the number of erroneous states that are repairable. Thus, with this measure, one can determine between several versions of a given circuit, which one is more robust (see gcd example, section 6.1). In a more general way, it can be used to determine a minimal set of registers to be protected for guaranteeing a recovery (see itc-b03 and CAN-Bus interface examples, in sections 6.2 and 6.3). Even for non robust circuits, we bring out information since the potential recovering capability is computed in addition to the eventual one. Our second metric evaluates the velocity of circuits to recover by computing the lengths of elementary repairing sequences. The distribution of these sequences on the different lengths yields more precision about the way erroneous states are repaired (see gcd example).

Our model of robustness relaxes the strict conformance to a golden model which is generally adopted. Such a conformance is the highest level of robustness a circuit may conform, however this often leads to harden a lot of registers to obtain it (e.g. add protection by replication and vote). In many applications (video stream, loose synchronization), weaker robustness conformance are acceptable, like “guaranteeing a recovery into a safe state after some bounded sequence”. Our reparation model allows for characterizing the way the system recover by restricting the repairing sequences to conform to the reparation automaton.

A first perspective consists in refining this reparation model. In the presented study, the repairing sequences are selected by reaching safe states, simply defined by

the reparation automaton. We plan to consider more information into account, about the circuit execution or its environmental context. Indeed, the robustness requirement a circuit has to achieve greatly depends on its environment. In some cases, the so-called safe-configurations to be reached depend on the past of the error state, and this is not captured by the reparation automaton.

In its present state, the proposed approach opens interesting questions to be addressed to extend the size of circuits to be analyzed. The computation of robustness measures we propose is implemented in a BDD- and SAT- model-checking framework ; as it can be seen in the Experiment Section, measure computation is very efficient for small sized circuit (up to 20 – 50 flip-flops) but the combinatorial blow up is quickly reached. The computation times of the Emitter and Receiver stations of the CAN bus interface illustrates this problem, and a compositional reasoning has to be adopted to combine the measures of internal components. Indeed, the 100% robustness of two components does not guarantee the 100% robustness of their product, other synchronization properties have to be added. Another way to analyze complex systems concerns the use of abstractions, however one has to carefully define the reparation automaton of the abstracted model in order to ensure the automatic transposition of 100% robustness result from the abstract model to the concrete one. These two points will be subjects of forthcoming research.

References

1. J. Abke, E. Bohl, and C. Henno. Emulation based real time testing of automotive applications. In *Proc. of 4th IEEE international On-Line Testing Workshop*, pages 28–31, 1998.
2. S. Baarir, C. Braunstein, R. Clavel, E. Encrenaz, J.-M. Ilié, R. Leveugle, I. Mounier, L. Pierre, and D. Poitrenaud. Complementary formal approaches for dependability analysis. In *Proc. International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 331–339. IEEE Computer Society, 2009.
3. H. Bidgoli. *Handbook of information security, volume 3*. Wiley, 2006.
4. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.
5. E. Birnbaum and E. L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, 10:457–477, 1999.
6. A. Church. Application of recursive arithmetic to the problem of circuit synthesis. In *Summaries of talks presented at the Summer Institute for Symbolic Logic*, pages 3–50, Princeton, 1957. Communications Research Division, Institute for Defense Analyses.
7. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
8. F. Corno, M. Sonza Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results. *IEEE Design and Test of Computers*, 17(3):44–53, 2000.
9. L. Naviner et J. F. Naviner D. Franco, M. Vasconcelos. Signal probability for reliability evaluation of logic circuits. *Microelectronics Reliability Journal*, 48, 2008.
10. J.-M. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche. An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip. In *IEEE RPS-CDR 47th annu. int. Reliability Physics symp.*, Montreal, Canada, 2009. IEEE.
11. R.I. Davis, A. Burns, R. Bril, and J. Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time systems*, 35(3):239–272, 2007.
12. E. W. Dijkstra. Self-stabilization systems in spite of distributed control. *Communication of the ACM*, 17:643–644, 1974.

13. S. Dolev. *Self-Stabilization*. The MIT Press, 2000.
14. G. Fey and R. Drechsler. A basis for formal robustness checking. In *Proc. IEEE International Symposium on Quality Electronic Design*, pages 784–789. IEEE Computer Society, 2008.
15. G. Fey, A. Sülflow, and R. Drechsler. Computing bounds for fault tolerance using formal techniques. In *Proc. Design Automation Conference*, pages 190–195. ACM, 2009.
16. L. Naviner et J. F. Naviner G. Goncalves dos Santos Jr, E. Crespo Marques. Using error tolerance of target application for efficient reliability improvement of digital circuits. *Microelectronics Reliability Journal*, 50(9-11), 2010.
17. M. Garcia-Valderas, M. Portela-Garcia, C. Lopez-Ongil, and L. Entrena. Extensive seu impact analysis of a pic microprocessor for selective hardening. *IEEE Trans. on Nuclear Science*, 57(4), 2010.
18. The VIS group. VIS : A system for verification and synthesis. In *Proc. International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432. Springer-Verlag, 1996.
19. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
20. E. Jenn, J. Arlat, R. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into VHDL models: the MEPHISTO tool. In *Proc. of 24th symposium on Fault-Tolerant Computing (FTCS)*, pages 66–75, 1994.
21. U. Krautz, M. Pflanz, C. Jacobi, H. Tast, K. Weber, and H. Vierhaus. Evaluating coverage of error detection logic for soft errors using formal methods. In *Proc. Conference on Design, Automation and Test in Europe*, pages 176–181. European Design and Automation Association, 2006.
22. S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. Hayes. Probabilistic transfert matrices in symbolic reliability analysis of logic circuits for reliability evaluation of logic circuits. *Trans. on Design Automation of Electronic Systems (TODAES)*, 13(1), 2008.
23. R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutation. In *Proc. IEEE International On-Line Testing Symposium*, pages 260–265. IEEE Computer Society, 2005.
24. R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *Proc. of Design, Automation and Test in Europe Conference (DATE)*, pages 502–506, 2009.
25. R. Leveugle and K. Hadjiat. Multi-level fault injections in VHDL descriptions: Alternative approaches and experiments. *Journal of Electronic Testing: Theory and Applications*, 19(5):559–575, 2003.
26. C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena. Autonomous fast emulation: A new fpga-based acceleration system for hardness evaluation. *IEEE Trans. on Nuclear Science*, 54(1), 2007.
27. Ilia Polian, John P. Hayes, Sudhakar M. Reddy, and Bernd Becker. Modeling and mitigating transient errors in logic circuits. *IEEE Transactions on Dependable and Secure Computing*, 99(PrePrints), 2010.
28. S. Seshia, W. Li, and S. Mitra. Verification-guided soft error resilience. In *Proc. Conference on Design, Automation and Test in Europe*, pages 1442–1447. EDA Consortium, 2007.
29. O. Shtrichman. Tuning SAT checkers for bounded model checking. In *Proc. International Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2000.
30. ISO technical commitees TC22/SC3. *Road Vehicles – Controller Area Networks – parts 1 to 5*. International Organization for Standardization, 2003.
31. M. Thurley. sharpSAT - counting models with advanced component caching and implicit BCP. In *Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429, Seattle, WA, USA, August 2006. Springer.