electronics

MDPI

Article

Techniques for SAT-Based Boolean Reasoning on Multiple Faults Affecting Logic Cells and Interconnects in Digital ICs

Francesco Dall'Occo [1], Marcello Dalpasso [2] and Michele Favalli [1,*]

[1] Department of Engineering, University of Ferrara, Viale Saragat, 2, 44122 Ferrara, Italy; francesco.dallocco@unife.it

[2] Department of Information Engineering, University of Padova, Viale Gradenigo, 6A, 35131 Padova, Italy; marcello.dalpasso@unipd.it

* Correspondence: michele.favalli@unife.it

Abstract: We propose a new method for SAT-based Boolean reasoning on multiple defects in digital ICs. Although it does not explicitly consider a specific fault model such as model-based techniques, it allows us to consider more realistic cases than model-free approaches. In particular, it can be used to account for (a) faults resulting in monotonic errors at the output of a cell and (b) faults, such as breaks or bridgings, that may corrupt the propagation of a signal from its fan-out branches. The model can be used for either standard gates or more complex combinational modules. Examples are shown for applications requiring the consideration of multiple defects such as fault diagnosis and reliability analysis. The feasibility of the proposed approach is assessed by results on a set of combinational benchmarks.

Keywords: digital integrated circuits; multiple faults; fault diagnosis; reliability analysis; Boolean satisfiability

check for updates

Citation: Dall'Occo, F.; Dalpasso, M.; Favalli, M. Techniques for SAT-Based Boolean Reasoning on Multiple Faults Affecting Logic Cells and Interconnects in Digital ICs. *Electronics* **2022**, *11*, 382. https://doi.org/10.3390/electronics11030382

Academic Editor: Arturo de la Escalera Hueso

Received: 30 December 2021

Accepted: 25 January 2022

Published: 27 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The expected reliability reduction [1,2] in digital FET-based nano-circuits will hardly be managed by the classical single-fault paradigm widely used in testing and in some fault-tolerant design techniques. In several applications, multiple faults cannot be neglected during the design, test and reliability assessment processes, and the problem is expected to worsen in the perspective successors of such technologies.

The need for design automation tools accounting for multiple faults may be satisfied by either probabilistic or deterministic techniques, according to the application: probabilistic techniques [3] can estimate error probabilities, while deterministic ones can assess fault-tolerant capabilities with respect to given fault multiplicities.

In the deterministic case, Boolean proof engines have been successfully used in the synthesis, verification and testing of digital ICs. In the testing context, Boolean satisfiability (SAT) [4] has been used in test generation [5,6] and fault diagnosis [7,8] for several kinds of fault models, mainly considering single faults. Recently, SAT-based test generation has been applied to the case of multiple faults in combinational logic modules, showing that test sequences featuring full coverage on single stuck-at faults need very few additional test vectors to achieve full coverage on multiple gate-output stuck-at faults as well [9,10].

SAT-based fault diagnosis for single and multiple faults typically uses additional variables to encode the set of possible faulty circuits. In this regard, two possibilities have been explored depending on whether such additional variables are used to encode specific faulty behaviors of a logic cell (such as those induced by stuck-ats or other kind of faults) or to simply encode fault locations in a fault model-free approach [11].

In other cases, including fault-tolerant systems featuring either error detection or fault masking properties, the perspective is slightly different, but multiple faults still have to be accounted for. For instance, the error masking properties of a fault-tolerant system or the

accuracy of an approximate computing circuit may have to be assessed with respect to a given fault cardinality. In such cases, it may be required to reason about circuit properties in the presence of multiple faults.

The approaches that consider specific fault models mainly work at a structural level: they inject faults in all logic cells of the fault-free circuit by using extra logic gates and additional signals, thus modeling the circuit in the possible presence of multiple faults; then, they derive a CNF (i.e., conjunctive normal form) from such a model.

On the contrary, the model-free approaches [11] consider possible fault locations instead of specific faults. They can also directly modify the CNF representing the fault-free circuit by adding variables to the CNF that allow signals to assume values that are not consistent with the fault-free case.

In the case of a combinational circuit, the task of fault diagnosis is to find configurations of additional variables justifying a set of possibly erroneous output configurations as they are individuated by production testing. When analyzing the reliability of a circuit design, these techniques can be used to check whether multiple faults exist that produce output errors violating some required property.

In both cases, a huge set of additional variable assignments exists that satisfies the CNF of the circuit under diagnosis/reliability constraints. Therefore, cardinality constraints are necessary to achieve meaningful solutions. For instance, in [11], given a set of input assignments and output responses, a Quantified Boolean Formula (QBF) is built over the CNF describing the possible faulty circuits and it is solved under cardinality constraints.

In this context, we first compare model-based (considering stuck-at faults) approaches with model-free ones and we discuss their limitations. Then, we extend the model-free approach used in [11] in two ways. In the first one, we build the CNF describing the faulty circuits that account in a more realistic way for faults affecting logic cells while still avoiding structural modifications to the circuit. Then, a model is proposed that allows us to consider faults that cannot be accounted for by either existing structural or model-free fault injection techniques. In particular, these are faults affecting the circuit interconnections [12] or internal bridging faults resulting in intermediate voltages at the outputs of gates [13].

The problem of fault multiplicity is here approached using a Pseudo Boolean SAT solver/optimizer [14] to minimize the number of additional variables to be set in order to satisfy a given condition on the output error. In different applications, instead, fault multiplicity can be constrained to be smaller than a given threshold, while using the Pseudo Boolean SAT solver/optimizer to maximize, for instance, the number of output errors or their numerical value (in arithmetic units). Finally, a simpler SAT solver [15] can be used when optimization is not required.

The proposed method is not applied here to a specific problem of testing, diagnosis or design of fault-tolerant circuits: we performed some experiments regarding the fault multiplicity with respect to output error cardinality in an attempt to produce worst-case conditions for the solver. We used a set of relevant circuits including combinational benchmarks and some toy feed-forward full-connectivity neural networks using integer data. Results show the computational feasibility of the proposed approaches.

The main contributions of the proposed approach to the state of the art are (1) the definition of new techniques for multiple fault injection that are halfway between fault model-based approaches and model-free approaches, attempting to trade off the compactness and broad scope of the latter with the accuracy of the former; (2) the application to monotonic faults and faults affecting fan-out stems, which were never considered in this context; and (3) the exploitation of such techniques by Boolean reasoning techniques such as Boolean Satisfiability and Pseudo-Boolean Satisfiability in fault diagnosis and error analysis contexts.

This paper is organized in the following way. Section 2 briefly reviews and compares model-based structural and model-free approaches to multiple fault injection in digital circuits. In Section 3, we propose model-free approaches that attempt to describe the possible outcomes of realistic defects such as internal cell faults and break/opens in the

interconnects. The results achieved by means of the proposed approaches for a wide range of possible applications are described in Section 4 and compared to those achieved by model-based structural approaches. Finally, conclusions are drawn in Section 5.

2. Approaches to Multiple-Fault Injection

In this section, we briefly review model-based structural and model-free approaches to multiple fault injection.

2.1. Model-Based Structural Approaches

In structural approaches, the presence of multiple faults is typically modeled by adding extra logic to each cell (either a gate or a macro-cell) of the circuit.

For instance, a multiplexer may be used to select between the module output signal and a faulty value on the basis of selection variables that encode the presence/absence of stuck-at faults affecting such a signal (Figure 1a). By applying logic simplification and replacing the multiplexer with two gates, an equivalent scheme similar to that used in [9] can be obtained, as in Figure 1b.

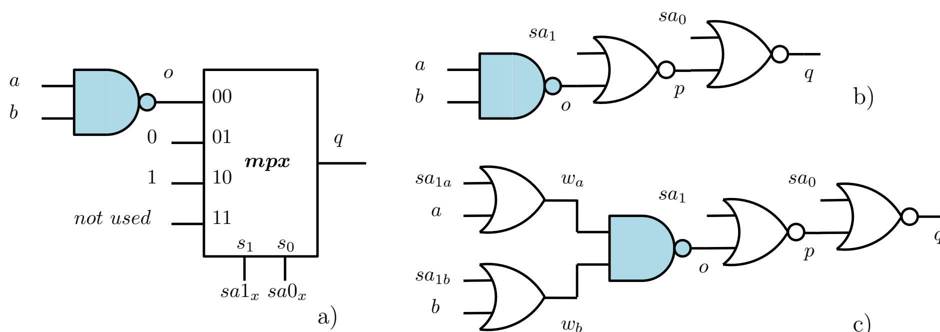

Figure 1. Examples of gate-level fault-injection techniques at the output of the shaded NAND gate with a multiplexer (**a**) or a simplified network (**b**). Subfigure (**c**) represents the model for the same NAND gate with faults at gate inputs and output.

Such a modified circuit model implicitly describes all possible multiple stuck-ats with no need to explicitly enumerate them as in a simulation. It can be used to build a miter comparing its outputs to those of the fault-free circuit. Some network possibly implementing a cardinality (A) constraint on the fault multiplicity can be added to such a miter. Different fault models can be modeled in a similar way.

In SAT-based approaches, a CNF of this circuit (or of the whole miter) is then built using the Tseitin [16] or other transformations.

Let us now consider a combinational logic cell with output y and local support $\boldsymbol{x}$. Under fault-free conditions, let f be the Boolean function implemented by the cell; i.e., $y = f(\boldsymbol{x})$. In order to model a fault affecting such a cell, y is replaced in its fan-out by a signal w implementing the function g; i.e., $w = g(\boldsymbol{x}, \boldsymbol{\alpha})$, where $\boldsymbol{\alpha}$ denotes the additional signals used to impose either the fault-free value or different faulty behaviors.

For instance, in Figure 1a,b, $\boldsymbol{x} = \{a, b\}$, $y = o$, $w = q$ and $\boldsymbol{\alpha} = \{sa_1, sa_0\}$, while the fault-free output function is

$$o = \neg(a \wedge b) \tag{1}$$

In Figure 1a, a multiplexer is used to inject a stuck-at fault at the gate output; therefore, o is replaced by q:

$$q = (\neg sa_1 \wedge \neg sa_0 \wedge o) \vee (sa_1 \wedge \neg sa_0)\,. \tag{2}$$

Finally, in Figure 1b,

$$q = \neg(\neg(sa_1 \vee o) \vee sa_0) \tag{3}$$

An additional constraint can impose that $(\neg sa_1 \vee \neg sa_0)$ holds true, thus avoiding meaninglkess configurations with two faults affecting the same signal.

When using Boolean satisfiability to reason on such a network, a CNF describing all its consistent signal assignments has to be derived.

In Figure 1b, such a CNF is

$$\begin{aligned}\Phi_0 = (a \vee o) \wedge (b \vee o) \wedge (\neg a \vee \neg b \vee \neg o) \wedge \\ (\neg o \vee \neg p) \wedge (\neg sa_1 \vee \neg p) \wedge (o \vee sa_1 \vee \neg p) \wedge \\ (\neg p \vee \neg q) \wedge (\neg sa_0 \vee \neg q) \wedge (p \vee sa_0 \vee \neg q)\end{aligned} \quad (4)$$

A more complete model that accounts also for input faults is illustrated in Figure 1c, where a modified NAND gate accounts for input stuck-at-1 faults as well. It describes all the possible stuck-at fault equivalence classes in a NAND gate. The resulting CNF is

$$\begin{aligned}\Phi_1 = (\neg a \vee w_a) \wedge (\neg sa_{1a} \vee w_a) \wedge (a \vee sa_{1a} \vee \neg w_a) \wedge \\ (\neg b \vee w_b) \wedge (\neg sa_{1b} \vee w_b) \wedge (b \vee sa_{1b} \vee \neg w_b) \wedge \\ (w_a \vee x) \wedge (w_b \vee x) \wedge (\neg w_a \vee \neg w_b \vee \neg o) \wedge \\ (\neg o \vee \neg p) \wedge (\neg sa_1 \vee \neg p) \wedge (o \vee sa_1 \vee \neg p) \wedge \\ (\neg p \vee \neg q) \wedge (\neg sa_0 \vee \neg q) \wedge (p \vee sa_0 \vee \neg q)\end{aligned} \quad (5)$$

The applications of structural fault injection to fault diagnosis or to the analysis of fault-tolerant properties aim at directly identifying specific faults within the circuit. Consider, for instance, the case of multiple-fault diagnosis: a culprit for a given input/output sequence is a configuration of the set of all the $\boldsymbol{\alpha}$ variables of circuit gates that is consistent with the considered input/output sequence.

2.2. Model-Free Approaches

If, in contrast to model-based approaches, $\boldsymbol{\alpha}$ variables are free to vary for the same gate in different circuit copies (similarly to [7]), we have a fault-model-independent diagnosis, which does not provide a fault dictionary based on stuck-ats (or other modeled faults), but a list of possible fault locations. For instance, in the model of Figure 1b, it may be possible that some input/output configuration is consistent with $sa_1 \wedge \neg sa_0$, while others are consistent with $\neg sa_1 \wedge sa_0$.

In [11], this problem has been approached by initially considering a structural injection scheme similar to that in Figure 1a. In particular, a two-way multiplexer is placed at the output o of a cell, while the other data input is v and the selection signal is ξ, thus resulting in the following Equation:

$$q = (o \wedge \neg\xi) \vee (v \wedge \xi) \quad (6)$$

where q is the new output. Differently from the structural injection scheme in Figure 1, the input v is allowed to vary within a diagnosis set of input and output configurations. In fact, in the QBF formulation of the diagnosis problem considered in [11], such a variable is not bounded, thus allowing for the gate output to have either a faulty or a fault-free value.

Such a description was used in [11] to derive an injection methodology that allows us to avoid the use of structural modifications and the subsequent derivation of a CNF formula. The fault is implicitly injected within the CNF formula with savings in the number of variables and clauses, while the CNF still represents the possible circuit behaviors in the presence of multiple faults.

In particular, the fault-free CNF of each cell is considered in turn, and a variable ξ is added to all the clauses of such a CNF in order to let the input variables and the output variable be assigned to values that are not consistent with the fault-free function of the cell; then, the clause may still be true by setting ξ as true.

As an example, consider, again, the case of a NAND gate ($o = \neg(a \wedge b)$) whose CNF describing the consistent assignments to a, b and o:

$$\Phi_{nand} = (a \vee o) \wedge (b \vee o) \wedge (\neg a \vee \neg b \vee \neg o) \quad (7)$$

is rewritten as:

$$\Phi_{nand}^{faulty} = (a \vee o \vee \xi) \wedge (b \vee o \vee \xi) \wedge (\neg a \vee \neg b \vee \neg o \vee \xi) \quad (8)$$

This formula allows for faulty assignments to o, such as, for instance, $a = 0$ and $o = 0$, that are justified by setting $\xi = 1$. It is evident that this new CNF may be satisfied by any faulty behavior, while, in contrast to structural approaches, the number of clauses does not increase with respect to the fault-free case. Note that, in this case, $\boldsymbol{\alpha} = \{\xi\}$.

2.3. Comparison

In this section, we compare structural and model-free approaches to multiple fault injection by considering a circuit affected by a resistive bridging fault.

In particular, we suppose that a three-input NAND FET cell is affected by an internal resistive bridging fault (Figure 2a). The detection of this kind of faults depends on circuit and bridging parameters that determine the faulty output voltage of o. In this example, it is supposed that the fault results in an output error (i.e., a low voltage) when applying input vectors $abc = 101$ and 110, while the test 100 does not produce an error because of the increased pull-up conductance. Let us now consider two possible CNFs for the cell: the former corresponds to all the possible input and output stuck-at faults (Figure 2b), while the latter is a model-free one as in Equation (8).

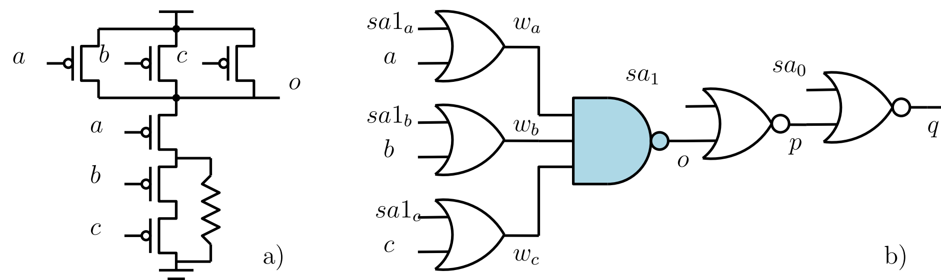

Figure 2. (**a**) Three-input NAND cell affected by an internal bridging fault. (**b**) Structural injection of input and output stuck-at faults for the same cell.

In the structural case accounting for input and output stuck-ats, the CNF is

$$\begin{aligned}\Phi_s = {} & (\neg a \vee w_a) \wedge (\neg sa_{1a} \vee w_a) \wedge (a \vee sa_{1a} \vee \neg w_a) \wedge \\ & (\neg b \vee w_b) \wedge (\neg sa_{1b} \vee w_b) \wedge (b \vee sa_{1b} \vee \neg w_b) \wedge \\ & (\neg c \vee w_c) \wedge (\neg sa_{1c} \vee w_c) \wedge (c \vee sa_{1c} \vee \neg w_c) \wedge \\ & (w_a \vee o) \wedge (w_b \vee o) \wedge (w_c \vee o) \wedge (\neg w_a \vee \neg w_b \vee \neg w_c \vee \neg o) \wedge \\ & (\neg o \vee \neg p) \wedge (\neg sa_1 \vee \neg p) \wedge (o \vee sa_1 \vee \neg p) \wedge \\ & (\neg p \vee \neg q) \wedge (\neg sa_0 \vee \neg q) \wedge (p \vee sa_0 \vee \neg q)\end{aligned} \quad (9)$$

It can be verified that no assignment exists to the variables in $\boldsymbol{\alpha} = \{sa_{1a}, sa_{1b}, sa_{1c}, sa_1, sa_0\}$ that ensures that $\Phi_s = 1$ is satisfied for all values of the inputs and the output of the faulty cell, while for specific values of such variables, there is always a value of $\boldsymbol{\alpha}$ satisfying $\Phi_s = 1$.

In the model-free case, it is

$$\Phi_m = (a \vee o \vee \xi) \wedge (b \vee o \vee \xi) \wedge (c \vee o \vee \xi) \wedge (\neg a \vee \neg b \vee \neg c \vee \neg \vee \neg o \vee \xi) \quad (10)$$

If we set $\xi = 1$, $\Phi_m = 1$ for any assignment that is consistent with the faulty gate function.

Let us now suppose that Equations (9) and (10) are used in a simple fault diagnosis of a synchronous sequential circuit (Figure 3) where we injected the fault of Figure 2a in the shaded cell. The figure shows the output values produced by such a faulty cell as they

are computed by fault simulation after injection in all time frames. Then, we suppose that, in order to justify such output values, a SAT-based diagnosis tool is applied to the CNF of the circuit in the structural or the model-free approach. In this simple experiment, the only task of SAT is to justify the output values for all time frames, given the input. This can be performed by setting suitable values for the variables in $\boldsymbol{\alpha}$.

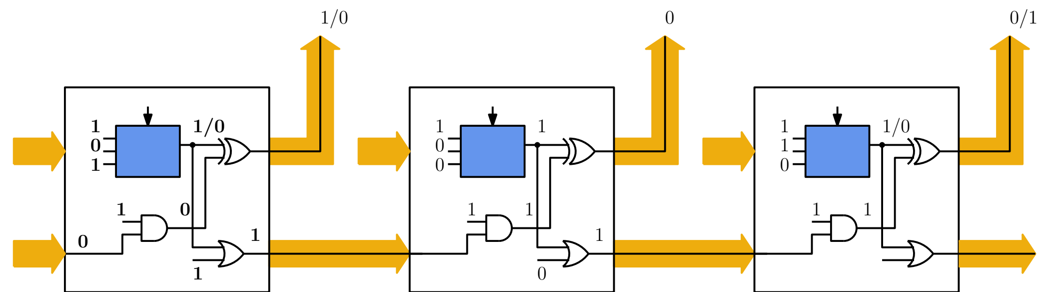

Figure 3. Time-frame expansion of a faulty synchronous sequential circuit featuring a cell with an injected fault.

In the stuck-at based case, no single configuration of $\boldsymbol{\alpha}$ may produce the same faulty behavior for all time frames. On the contrary, based on Equation (9), it is easy to verify that, if different values of such variables are allowed for each time frame, the SAT solver can justify the states of the circuit. Conversely, if the faulty circuit is described by Equation (10), a single value for ξ can be used for all frames.

3. Accounting for Realistic Defects

The model-free approach in Equation (8) implicitly describes any possible kind of faulty behavior of the gate, including not physically realistic ones (for instance, an XOR-like behavior of a faulty NAND gate). This may result in unnecessarily wide fault location sets in the case of fault diagnosis or in some pessimism in reliability analysis.

To approach this problem, while still not using a structural approach encoding all possible faulty behaviors of a cell, we do not use a single additional variable for all the clauses of a cell, but one variable (ξ_0) to be added to clauses featuring the true form literal of the cell output and another one (ξ_1) to clauses featuring the false form literal of the cell output.

In this way, we can describe all faults that results in a monotonic error at the output of a cell: only the on-set but not the off-set of its function, or vice versa, can be affected by the fault. In the case of a NAND gate, this results in

$$\Phi = (a \vee o \vee \xi_0) \wedge (b \vee o \vee \xi_0) \wedge (\neg a \vee \neg b \vee \neg o \vee \xi_1) \tag{11}$$

If we do not allow both ξ_0 and ξ_1 to be true at the same time, the set of modeled faulty behaviors is restricted to the more realistic cases including internal bridging faults [13]. This model can be easily generalized to any kind of cell.

Example

Let us consider again a fault diagnosis experiment over a simple faulty combinational circuit that contains two three-input NAND gates whose outputs are XORed. One of these NANDs is supposed to be affected by the same fault as in Figure 2a.

Figure 4 shows the cell inputs and the corresponding output values with two input vectors. Let us first suppose that the CNFs of the two cells are described by Equation (10):

$$\begin{aligned} \Phi_p = {} & (a \vee o \vee \xi^p) \wedge (b \vee o \vee \xi^p) \wedge (c \vee o \vee \xi^p) \wedge \\ & (\neg a \vee \neg b \vee \neg c \vee \neg p \vee \xi^p) \\ \Phi_q = {} & (a \vee o \vee \xi^q) \wedge (b \vee o \vee \xi^q) \wedge (c \vee o \vee \xi^q) \wedge \\ & (\neg a \vee \neg b \vee \neg c \vee \neg q \vee \xi^q) \end{aligned} \tag{12}$$

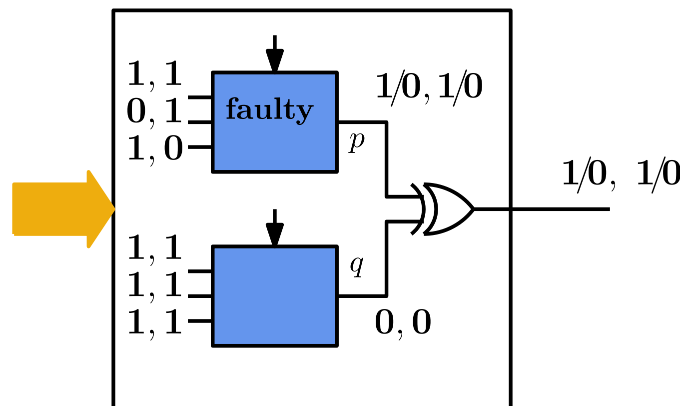

Figure 4. Combinational circuit featuring a faulty cell. Two possible input vectors are shown separated by commas. The notation 1/0 denotes that the considered signal has the values 1 and 0 in the fault-free and in the faulty circuit, respectively.

It is easy to verify that a SAT solver (working with the cardinality constraint $A < 2$) succeeds in justifying the erroneous output value by setting either $\xi^p = 1$ or $\xi^q = 1$, thus meaning that two possible fault locations have to be considered.

Conversely, if we use the proposed approach, the CNFs are

$$\begin{aligned}\Phi_p = {} & (a \vee o \vee \xi_0^p) \wedge (b \vee o \vee \xi_0^p) \wedge (c \vee o \vee \xi_0^p) \wedge \\ & (\neg a \vee \neg b \vee \neg c \vee \neg \xi_1^p) \\ \Phi_q = {} & (a \vee o \vee \xi_0^q) \wedge (b \vee o \vee \xi_0^q) \wedge (c \vee o \vee \xi_0^q) \wedge \\ & (\neg a \vee \neg b \vee \neg c \vee \neg \xi_1^q)\end{aligned} \tag{13}$$

In this case, the SAT solver will provide only ξ_0^q as a solution, thus restricting the list of possible culprits.

Other problems that cannot be accounted for by the model in Equation (8) are related either to faults producing output intermediate voltages such as transistor stuck-ons or internal resistive bridging faults [13] or to faults affecting interconnections, such as breaks and opens [12].

In the former case, the gates in the fan-out of the faulty cell may interpret its output faulty intermediate voltage in different ways depending on their actual logic thresholds.

In the latter case, a break or open affecting some position of the layout of an interconnection may cause the gate output value to correctly propagate to a subset only of fan-out gates. This case is shown in Figure 5 where, with the applied inputs, it is supposed that, because of a break affecting the fan-out interconnects of gate q, the gates s and v receive an incorrect input logic value. The same behavior may occur because of a resistive bridging affecting gate q and result in its output signal having an intermediate voltage value that is supposed to be interpreted as low by gate r and as high by gates s and v because of statistical variations of the logic threshold of such gates.

In the case of fault diagnosis, a model-free approach cannot justify this condition with a fault cardinality equal to 1, while with a cardinality equal to 2, it would identify s and v as the possible fault locations (or any combination of gates in their transitive fan-out in more complex circuits).

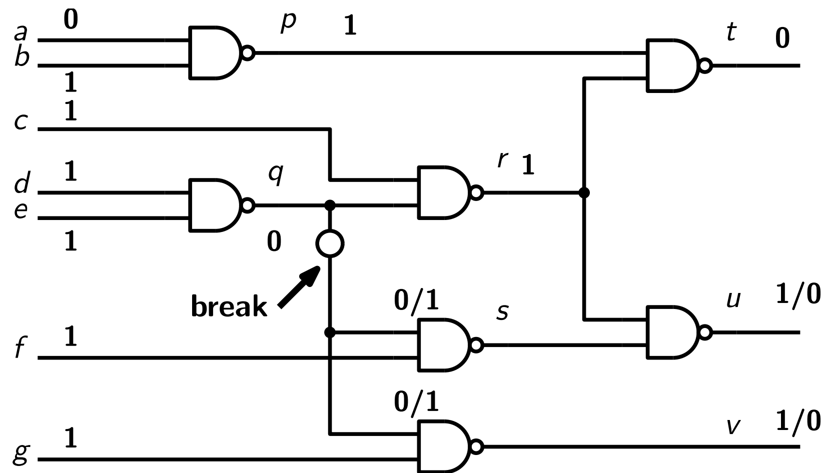

Figure 5. Example showing a broken interconnection that results in logic errors in two fan-out branches out of three.

To approach this problem while keeping low values of fault cardinality, we extend fault injection in the CNF by using a different additional variable in each clause of the CNF of a gate, thus providing the following CNF in the case of a NAND gate. The starting point is still Equation (8), which, in the hypothesis that a and b are fan-out-stems, is rewritten as

$$\Phi = (a \vee o \vee \xi^{ao}) \wedge (b \vee o \vee \xi^{bo}) \wedge (\neg a \vee \neg b \vee \neg o \vee \xi^{ao} \vee \xi^{bo}) \tag{14}$$

If one or both inputs are not stem, some simplification can be made, while Equation (14) can be easily modified to account for unate faults:

$$\Phi = (a \vee o \vee \xi_0^{ao}) \wedge (b \vee o \vee \xi_0^{bo}) \wedge (\neg a \vee \neg b \vee \neg o \vee \xi_1^{ao} \vee \xi_1^{bo}) \tag{15}$$

By using such an approach, every gate input variable branching out from a fan-out stem is related to an additional variable that we use to describe the possible faults affecting the fan-out branches. Then, we add a stem-related variable (χ) equal to the disjunction of the additional variables related to its fan-out branches. In this way, faults affecting multiple fan-out branches can be reduced to a single fault affecting the fan-out stem.

In order to describe this approach, let us consider the CNF of the gates driven by the fan-out stems q and r in the example of Figure 5:

$$\begin{aligned}
\Phi_r &= (c \vee r \vee \xi^{cr}) \wedge (q \vee r \vee \xi^{qr}) \wedge (\neg c \vee \neg q \vee \neg r \vee \xi^{cr} \vee \xi^{qr}) \\
\Phi_s &= (q \vee s \vee \xi^{qs}) \wedge (f \vee s \vee \xi^{fs}) \wedge (\neg q \vee \neg f \vee \neg s \vee \xi^{qs} \vee \xi^{fs}) \\
\Phi_v &= (q \vee v \vee \xi^{qv}) \wedge (g \vee v \vee \xi^{gv}) \wedge (\neg q \vee \neg g \vee \neg v \vee \xi^{qv} \vee \xi^{gv}) \\
\Phi_t &= (p \vee t \vee \xi^{pt}) \wedge (r \vee t \vee \xi^{rt}) \wedge (\neg p \vee \neg r \vee \neg t \vee \xi^{pt} \vee \xi^{rt}) \\
\Phi_u &= (r \vee u \vee \xi^{ru}) \wedge (s \vee u \vee \xi^{su}) \wedge (\neg r \vee \neg s \vee \neg u \vee \xi^{ru} \vee \xi^{su})
\end{aligned} \tag{16}$$

The CNF of the circuit is completed by the CNFs describing gates p and q which use only one ξ variable because they do not have stems in their fan-in:

$$\begin{aligned}
\Phi_p &= (a \vee p \vee \xi^{p}) \wedge (b \vee p \vee \xi^{p}) \wedge (\neg b \vee \neg b \vee \neg p \vee \xi^{p}) \\
\Phi_q &= (d \vee q \vee \xi^{q}) \wedge (e \vee q \vee \xi^{q}) \wedge (\neg d \vee \neg e \vee \neg q \vee \xi^{q})
\end{aligned} \tag{17}$$

The effects of faults affecting the branches of a fan-out stem are then captured by an additional variable holding true if one of the ξ variables appearing in the clauses containing the stem variable is true. For instance, in the case of q, we add the variable χ^q as

$$\chi^q \leftrightarrow \xi^{qr} \vee \xi^{qs} \vee \xi^{qv} \tag{18}$$

The same is done for the fan-out stem r:

$$\chi^r \leftrightarrow \xi^{rt} \vee \xi^{ru} \tag{19}$$

Now, a cardinality constraint can be built as the following pseudo-Boolean formula (which can be translated to a CNF):

$$A = \chi^q + \chi^r + \xi^p + \xi^q \tag{20}$$

In this way, the CNF of the whole circuit constrained by the values of primary inputs (PIs) and primary outputs (POs) can be satisfied under the cardinality constraint $A = 1$ with a model where the additional variables χ^q, ξ_0^s and ξ_0^v are set true, while other additional variables are set false. Note that if we do not use such additional variables, the CNF can be satisfied only for $A = 2$.

Extension to Combinational Macros

Logic gates do not represent all the possible combinational cells in current and perspective technologies. For instance, this may be the case for the Look-Up Tables (LUT) used in the large majority of Field-Programmable Gate Arrays (FPGAs). This may hold also for perspective technologies featuring small nano-fabrics [17] or more complex logic gates [18]. In such cases, our approach cannot be directly applied to a Tseitin transform [4,16].

Let us suppose that a Disjunctive Normal Form (DNF) expression is available for both the module output and its complement:

$$y = \bigvee_i p_i, \qquad \neg y = \bigvee_j q_j \tag{21}$$

where p_i and q_j are product terms. Note that these function representations may often be available during the logic synthesis as a sub-product of two-level logic minimization for single cells. In this case, the CNF for the considered macro can be computed as

$$\begin{aligned} \Phi = \left(\bigwedge_i (p_i \rightarrow y)\right) \wedge \left(\bigwedge_j (q_j \rightarrow \neg y)\right) = \\ \left(\bigwedge_i (\neg p_i \vee y)\right) \wedge \left(\bigwedge_j (\neg q_j \vee \neg y)\right) = \\ \bigwedge_i \left(\bigvee_{l_{ik} \in p_i} \neg l_{ik} \vee y\right) \wedge \bigwedge_j \left(\bigvee_{l_{jk} \in q_j} \neg l_{jk} \vee \neg y\right) \end{aligned} \tag{22}$$

(using De Morgan's laws) where l_{ij} and l_{jk} are the literals of the products in the DNF expressions of the module output and of its complement, respectively. This fault-free cell model can be used to inject faults. For instance, if we consider monotonic faults, we can write

$$\Phi = \bigwedge_i \left(\bigvee_{l_{ik} \in p_i} \neg l_{ik} \vee y \vee \xi_0\right) \wedge \bigwedge_j \left(\bigvee_{l_{jk} \in q_j} \neg l_{jk} \vee \neg y \vee \xi_1\right) \tag{23}$$

Example

Consider a cell implementing the function $y = (a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c)$; hence, $\neg y = (a \wedge b) \vee (\neg a \wedge \neg b) \vee \neg c$. The fault-free CNF of such a cell is

$$\begin{aligned} \Phi_{cell} = (\neg a \vee b \vee \neg c \vee y) \wedge (a \vee \neg b \vee \neg c \vee y) \wedge \\ (\neg a \vee \neg b \vee \neg y) \wedge (a \vee b \vee \neg y) \wedge (c \vee \neg y) \end{aligned} \tag{24}$$

The resulting faulty CNF, which can be used to represent all monotonic faults within the considered module, is

$$\Phi_{cell}^{faulty} = (\neg a \vee b \vee \neg c \vee y \vee \xi_0) \wedge (a \vee \neg b \vee \neg c \vee y \vee \xi_0) \wedge \\ (\neg a \vee \neg b \vee \neg y \vee \xi_1) \wedge (a \vee b \vee \neg y \vee \xi_1) \wedge (c \vee \neg y \vee \xi_1) \quad (25)$$

For instance, this model accounts for a fault k resulting in a faulty output function equal to $(a \wedge \neg b \wedge c) \vee (\neg a \wedge c)$, which, in a regular array, may occur because of a missing crosspoint and is not equivalent to any stuck-at fault. For the sake of brevity, the extension of our model to multi-output cells is not discussed in this paper.

4. Results

The proposed approaches can be used to reason regarding several properties of digital ICs in the presence of multiple faults. Without addressing any specific application, we simply compare its efficiency with that of the structural injection schemes reported in Figure 1b,c. Columns 2–4 of Table 1 show the characteristics of the considered combinational benchmarks taken from the ISCAS85 [19] and ITC99 [20] sets as they are described at the gate level. Moreover, three toy feed-forward integer data neural networks have been added.

Starting from the netlist of each benchmark as it is described in BLIF [21], we have built the CNF of a miter featuring the conjunction of the fault-free CNF, the CNF describing the faulty circuit (sharing the same variables for PIs), and the CNF of a comparator fed by the POs' variables of the two copies of the circuit. The CNF of the faulty circuit can be built in different ways depending on the considered fault-injection model:

- When considering the model for monotonic faults, we use Equation (11) to modify the fault-free CNF;
- When considering structural models, we used a script to add extra gates to the benchmarks' fault-free netlist such as in Figure 1a,b, and then we converted it to a CNF.

The CPU time used to build the miter is negligible with respect to that spent by the SAT solver, and it is not considered.

We initially used the minisat SAT solver [15]. The experiments were performed on an Intel® Core™ i7-5500U CPU running at 2.4 GHz with 8 GB of memory under the Linux operating system.

The first experiment aimed at evaluating the computational effort required by a SAT solver in providing a model featuring a number of errors at POs above a given threshold in the presence of an upper bound on the number of internal faults using our approach (Equation (11)). In some cases, because of such an upper bound, the SAT solver may fail to justify an output error configuration. This problem is common to all the considered techniques and may occur if the number of faults is much lower than the number of output errors.

In particular, the miter's CNF is augmented by clauses that allow us to count the number of errors, E, and the number of $\boldsymbol{\alpha}$ variables at 1, A (which we consider as the fault cardinality). This allows us to set constraints on the minimum number of errors, E_{min}, and on the maximum number of faulty cells, A_{max}.

The larger E_{min} and the smaller A_{max} are, the larger the computational effort is expected to be, because we prevent the SAT solver from producing trivial solutions featuring a large number of faulty cells in the vicinity of POs. Such an experiment is only used to characterize the computational costs of the different modeling techniques in a way that is independent of the specific kind of application.

This is illustrated in Figure 6, which, for the benchmark b14, reports the CPU time as a function of E_{min}, with $A_{max} = 2$. The figure shows the results achieved by three considered methods. As expected, the CPU time increases with the minimum number of errors to be justified, even though the curves are non-monotonic because issues such as the circuit topology affect the execution of the SAT solver with different targets on errors. Moreover, the relevant computational cost of input/output fault injection can be noted.

Table 1. Characteristics of the considered benchmarks and performance of SAT-based reasoning on multiple faults for the three methods considered in this work. Data have been collected for $A_{max} = 2$ and $E_{min} = 20$; c is the number of clauses and CPU times are in seconds.

Benchmark	#PI	#PO	#Gates	Output s-at		Input/Output s-at		Our Method		*xy*-Decoding
Name				c	CPU Time	c	CPU Time	c	CPU Time	CPU Time
C3540 (unsat)	50	22	996	6887	0.17	10,135	1.62	2903	0.72	0.36
C5315	178	123	1457	10,090	0.44	14,833	1.65	4262	1.25	0.19
C6288	32	32	2416	16,880	0.58	28,304	2.39	7216	2.64	1.18
C7552	207	108	3510	23,692	0.67	31,036	1.74	9652	3.97	0.28
b14	5400	276	298	38,848	0.23	63,112	3.10	17,248	1.39	0.37
b15	498	519	7092	51,316	1.53	83,467	6.35	22,948	4.98	1.00
b17	1448	1509	23,092	167,804	10.80	271,661	41.36	75,436	45.85	9.28
b20	522	512	10,165	73,259	14.62	119,894	23.84	32,599	22.61	1.27
b21	522	512	9662	69,534	8.89	113,286	9.04	30,886	34.44	2.39
b22	1448	1509	23,092	107,421	3.46	174,636	1632.98	47,597	19.88	3.43
nn0 (3×3 neurons)	168	24	7440	78,126	541.34	112,878	1745.29	48,360	101.98	10.06
nn1 ($3 \times 4 \times 3$ neurons)	288	24	13,676	99,185	4090.34	171,523	too much	44,481	3203.90	82.18
nn2 ($3 \times 3 \times 3 \times 3$ neurons)	324	36	21,636	157,284	1746.00	274,008	too much	70,740	2203.11	48.58

The CPU time and the number of clauses (c) are shown in Table 1 for all benchmarks when considering (i) fault injection at gate output, (ii) stuck-at injection at gate inputs and output and (iii) our method. These data have been computed using $E_{min} = 20$ and $A_{max} = 2$.

As can be seen, the model featuring output stuck-ats and the proposed approach perform in a similar way (489 s on average compared to 462 s): such large values, however, are not suitable for several kinds of applications.

4.1. Cardinality of Faults

The analysis of the above-mentioned example shows that the cardinality constraint on the number A of faulty cells poses a relevant computational problem because of the size of the adder networks used to count α variables: in fault diagnosis or when verifying some fault-tolerant property of circuits, such cardinality constraints may be required. This worst-case scenario is common to all the considered techniques.

Any constraint on A can be expressed as a Pseudo Boolean (PB) one and can be translated to Boolean constraints by using a ones counter, which is a complex arithmetic network featuring several adder circuits; as an alternative, it can be obtained by using the PB-to-Boolean conversion embedded in a PB SAT solver such as minisat+ [14], which can use also combinational sorters which, however, do not provide benefits. The additional constraints due to the presence of such a network lead to relevant overheads in the time spent by the SAT solver.

To mitigate this problem, we used a simple approach based on a symbolic ordinate-based decoding of faults that, for $A_{max} = 2$, becomes an xy-decoding, used to illustrate this approach. We suppose that a faulty cell can be selected by the outputs of two decoders (x and y) and, as shown below, we perform this operation with no need to explicitly model the decoders. This operation works when considering single faults; in the case of multiple faults, we need to add pairs of decoders up to fill the maximum fault multiplicity. In this way, if n is the number of faulty cells, we reduce the size of the cardinality constraint from $O(n)$ to approximately $O(A_{max}\sqrt{n})$.

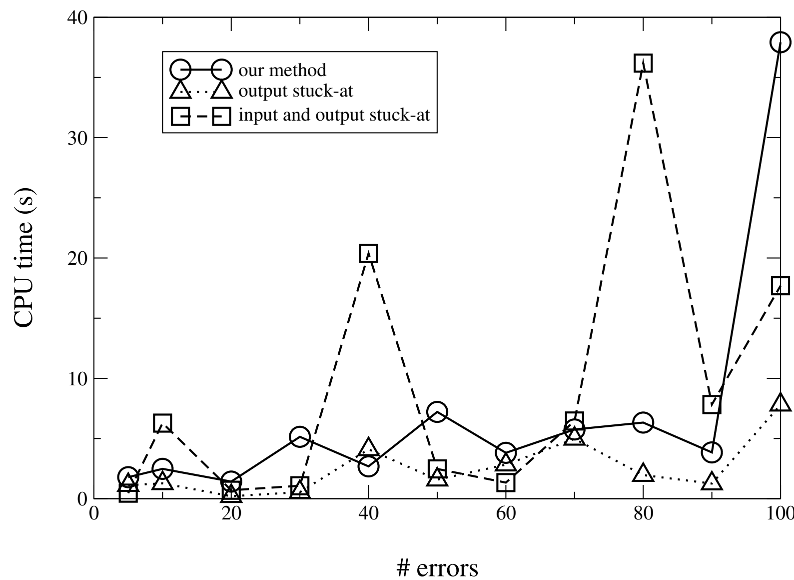

Figure 6. CPU time as a function of the minimum number of errors to be produced at POs for a gate-level implementation of the benchmark b14.

Such a strategy can be applied to any of the considered methods and is here instantiated only in our case. Consider the model describing monotonic errors, featuring two variables (ξ_0 and ξ_1) per cell and a maximum fault multiplicity for the whole circuit equal to 2. Then, for the i-th cell ($0 \leq i < n$), which is supposed to be mapped on ordinates (j, k) (for which approximately holds $0 \leq j, k \leq \sqrt{n}$),

$$\xi_0(i) \vee \xi_1(i) \rightarrow (X^1(j) \wedge Y^1(k)) \vee (X^2(j) \wedge Y^2(k)) \quad (26)$$

where $X^1(j)$ is the output of the x-decoder connected to such a cell and $Y^1(k)$ is the output of the y-decoder for the same cell. $X^2(j)$ and $Y^2(k)$ play the same role for a second decoder. The number of such signals is proportional to $\sqrt{n}$. Note that the pair of variables $(X^p(j), Y^p(k))$, $p \in \{1, 2\}$ is different for each cell of the circuit.

Now the (PB) cardinality constraint $A \leq A_{max}$ can be moved to the outputs of the decoder: the arithmetic sum of the outputs of each of them (that is $O(\sqrt{n})$) must be no larger than one.

If the solver needs to inject faults in two cells, two different pairs of (X, Y) variables will have their conjunction at 1; otherwise, if the fault has to be injected in one cell only, then either only one pair of variables has the conjunction at 1 or the two pairs corresponding to the same cell will be at 1.

Of course, by adding a suitable number of decoders, this technique can be generalized to any maximum cardinality in the number of faulty cells.

The results achieved by using this technique with our method are reported in the last column of Table 1, showing a remarkable speed-up (27 times) that results in an average CPU time equal to 17 s.

4.2. Application to Complex Modules

The approach described in Section 3 has been applied to an FPGA implementation of the benchmark b14 obtained by mapping such a circuit with abc [22] on four and five-input LUTs. In this regards, Figures 7 and 8 show the CPU time for $A_{max} = 2$ and different values of E_{min} for the four-input LUT and the five-input LUT, respectively. Each figure shows the CPU time for the injection of faults at cell output, cell inputs and output and our method.

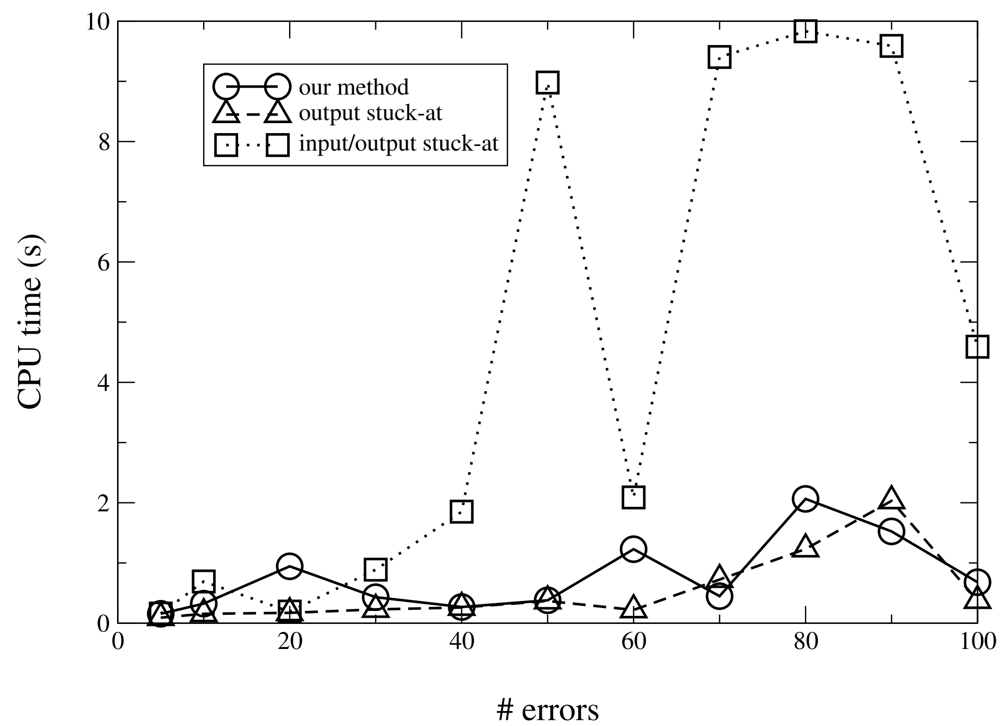

Figure 7. CPU time used by a SAT solver for processing the Boolean model of a four-input LUT implementation of b14 in the presence of multiple faults, for output stuck-ats, input/output stuck-ats and our method.

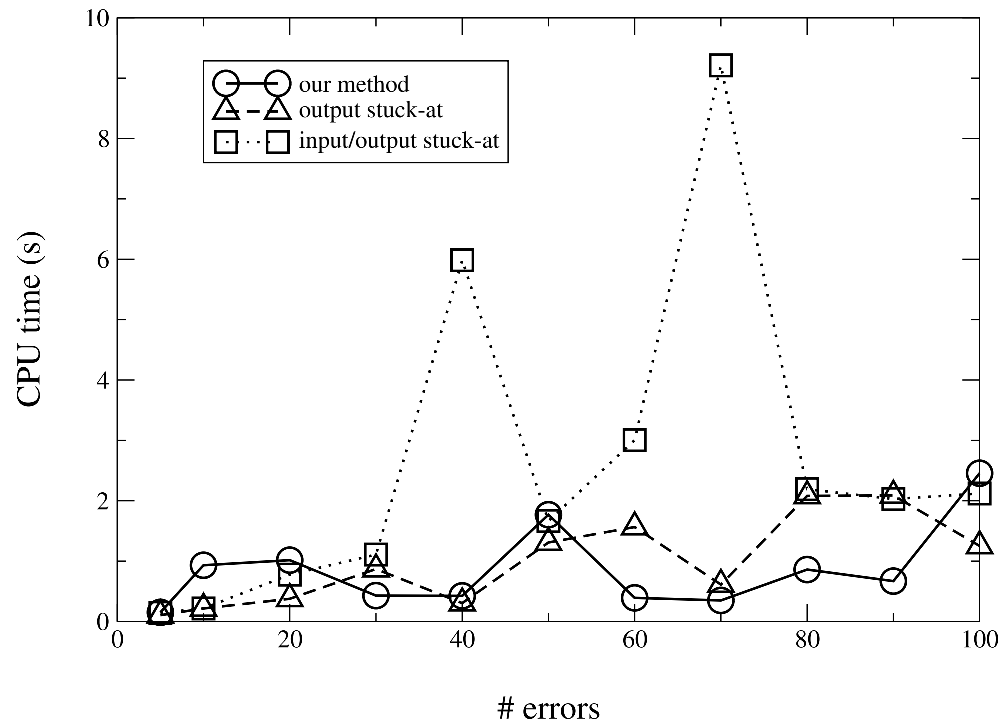

Figure 8. CPU time used by a SAT solver for processing the Boolean model of a five-input LUT implementation of b14 in the presence of multiple faults, for output stuck-ats, input/output stuck-ats and our method.

As can be seen, the CPU time, computed without using xy-decoding, is lower than in the gate-level case, because of the smaller total number of $\boldsymbol{\alpha}$ variables. Table 2 summarizes the results achieved for the set of considered benchmarks mapped on four-input and five-

input LUTs by considering the same conditions ($A_{max} = 2$, $E_{min} = 20$) used to obtain the data in Table 1.

Table 2. Number of cells for two different implementations of the set of considered benchmarks featuring four and five-input LUTs, respectively.

Name	#LUTs	4-Inputs LUT CPU Time (s)	#LUTs	5-Inputs LUT CPU Time (s)
C3540	356	0.04	282	0.24
C5315	481	0.23	361	0.18
C6288	515	0.36	635	0.48
C7552	508	0.24	434	0.23
b14	1861	1.21	1549	0.87
b15	3239	3.64	2649	3.36
b17	9892	16.76	7579	16.68
b20	3682	4.26	3024	3.70
b21	3732	4.35	3063	3.62
b22	4908	7.49	4016	5.07

The computational costs are smaller than in the gate-level case for all benchmarks.

4.3. Faults in Fan-Out Stems

As an application of the proposed method for faults in fan-out stems, we have considered a simple experiment partially emulating the operations of fault diagnosis. The target is to analyze the feasibility of the metho d outlined in Section 3 for faults resulting in breaks or intermediate voltages at fan-out stems.

To this purpose, we have initially used the c6288 benchmark that is a parallel multiplier. In our experiment, we have considered a set of possible diagnosis instances, each of which are constrained to present one of the possible (496) double errors at POs and a random input configuration.

For each instance, three different CNFs have been generated. The first one (to be referred to as case 1) uses the model in Equation (8) and a fault cardinality equal to 1. The second one uses the same model, but a cardinality equal to two (case 2). The third one, instead, uses the proposed model (Equation (14)) accounting for fan-out stems with a cardinality equal to 1 (case 3).

In case 1, only one cell may be faulty and the SAT solver is able to justify only 14 samples out of 496. In fact, even if several cells exist in the intersection of the transitive fan-ins of the two erroneous POs, an error in such cells may propagate also to other POs, thus resulting in an unsatisfiable instance. Of course, when considering a cardinality equal to two (case 2), we had 480 out of 496 satisfiable instances. The same result is obtained with the model considering fan-out stems (case 3), where we may have only one faulty stem.

The main difference between cases 2 and 3 is in the average number of possible solutions per instance, which are 354 and 129, respectively. From the point of view of fault diagnosis, this shows that the proposed technique (3) is capable of identifying faults possibly originated by a single defect. Conversely, by injecting faults at the cell level (2), additional solutions ($225 = 354 - 129$) are produced that correspond to multiple defects. From the point of view of the average CPU time spent by the solver, it is 0.47 s in case 1, while in cases 2 and 3, it is almost the same (0.81 s and 0.89 s, respectively).

This analysis has been performed also for the other benchmarks considered in Table 1. In some cases, because of the large number of outputs and, consequently, of possible double output errors, only a sample of 1000 of this kind of errors has been considered. The achieved results are summarized in Table 3, reporting for each method the percentage of satisfiable instances and the CPU time (computed using minisat+ with only combinational sorters as cardinality encoders). As can be seen, the CPU time in cases 2 and 3 is similar, thus showing that the additional χ variables do not give rise to relevant computational overheads. In addition, the table reports the number of solutions per sample provided by

case 2 and by our method. These last results show that, when considering defects affecting fan-out stems, we always achieve a reduction in the number of cases to be considered by fault diagnosis.

Table 3. Comparison between the model accounting for defects affecting fan-out stems (case 3) and models affecting cells with cardinality 1 (case 1) and 2 (case 2).

Bench	% of SAT Samples	(Case 1) CPU Time (s)	% of SAT Samples	Avg. No. of Solutions	(Case 2) CPU Time (s)	% of SAT Samples	Avg. No. of Solutions	(Case 3) Time (s)
C3540	3.6	1.95	63.0	108.0	0.16	62.4	95.1	0.17
C5315	0.6	0.45	93.6	497.5	0.48	91.0	404.0	0.49
C6288	2.8	0.47	100.0	354.0	0.81	96.8	129.0	0.89
C7552	0.8	0.48	97.0	117.5	0.86	83.0	108.0	0.96
b14	0.0	1.32	99.0	6.8	1.66	95.8	5.8	1.62
b15	0.8	1.53	100.0	16.6	1.88	92.4	14.8	3.62
b17	0.0	6.78	99.4	12.5	10.28	99.4	11.5	9.91
b20	0.0	2.48	98.1	11.3	2.94	98.1	9.7	3.01
b21	0.0	2.60	98.1	11.1	3.81	98.1	9.5	4.08
b22	0.0	2.87	98.1	20.1	4.67	98.1	18.0	4.70

4.4. Application to New Test and Verification Paradigms

The verification of fault-tolerant properties may require the determination of the minimal fault cardinality producing an output error. Moreover, the test generation for approximate computing may require unconventional approaches that generate tests maximizing some numerical difference between the faulty and the fault-free outputs. These are optimization problems that can be solved by PB SAT solvers/optimizers [14].

In the case of fault-tolerant circuits, we do not target specific applications, but we want to analyze the computational costs required to characterize worst-case scenarios from the point of view of errors when using the model described in Equation (11) for multiple faults resulting in monotonic errors. Therefore, we considered two general problems requiring the computation of (1) the maximum number of errors related to a given fault multiplicity and (2) the minimum number of faults producing a number of errors above a given threshold.

In case 1, at first, we considered the gate-level implementation of the benchmark b14 and we formulated the following PB optimization problem: maximize the number of output errors (E) for a given fault multiplicity $A \leq A_{max}$. Note that, in doing so, we did not exploit structural information. Figure 9 shows the results achieved by the PB solver/optimizer minisat+ using sorters to encode cardinality constraints (here, sorters work better than the adders used in the previous section).

The data achieved for the inverse problem (case 2) are shown in Figure 10, which reports the CPU time used to minimize the number of faulty cells in order to achieve a target number of errors. As can be seen, the CPU time scales in the worst way with respect to case 1 because the number of faulty cells is much larger than the number of possible errors.

Now, we use the simple feed-forward neural network nn0 to instantiate the cases, requiring us to quantify the error due to multiple faults from a numerical point of view. Such a network features three output words corresponding to three complement-two encoded integers. Let O_i^{ff}, $i = 1 \ldots 3$ be such output values in the fault-free case, while let O_i^f, $i = 1 \ldots 3$ be the corresponding values in the presence of multiple faults. In our experiment, we generated a CNF implicitly modeling a miter that describes (1) the fault-free circuit, (2) the faulty circuit (using our method) and (3) a comparison logic which, for each output word, checks for the accuracy of the result by comparing the absolute value of the

difference of the fault-free and faulty words to a constant (η_i). Each comparator produces an error signal e_i, $i = 1 \ldots 3$:

$$e_i = \begin{cases} 1 & \text{if } |O_i^{ff} - O_i^{f}| > \eta_i \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

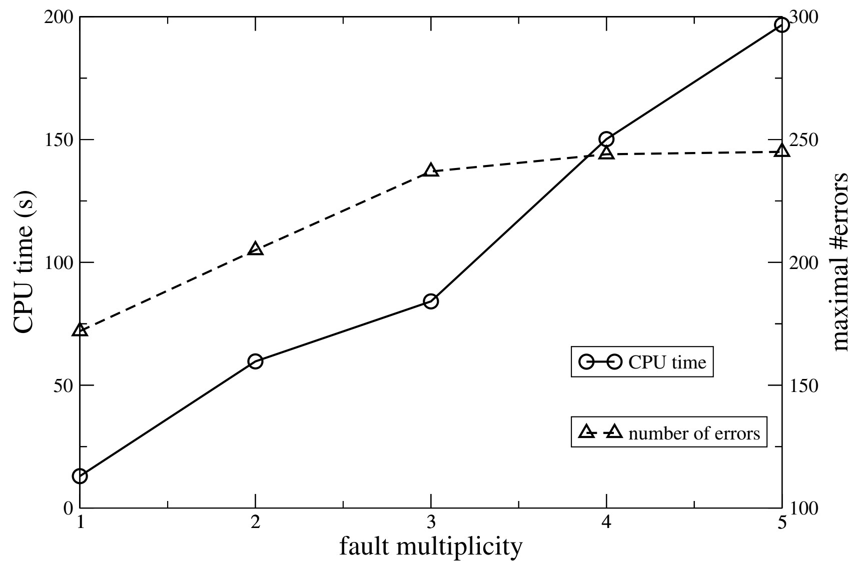

Figure 9. CPU time used by a PB-SAT optimizer to maximize the number of errors as a function of the maximal allowed number of faulty cells.

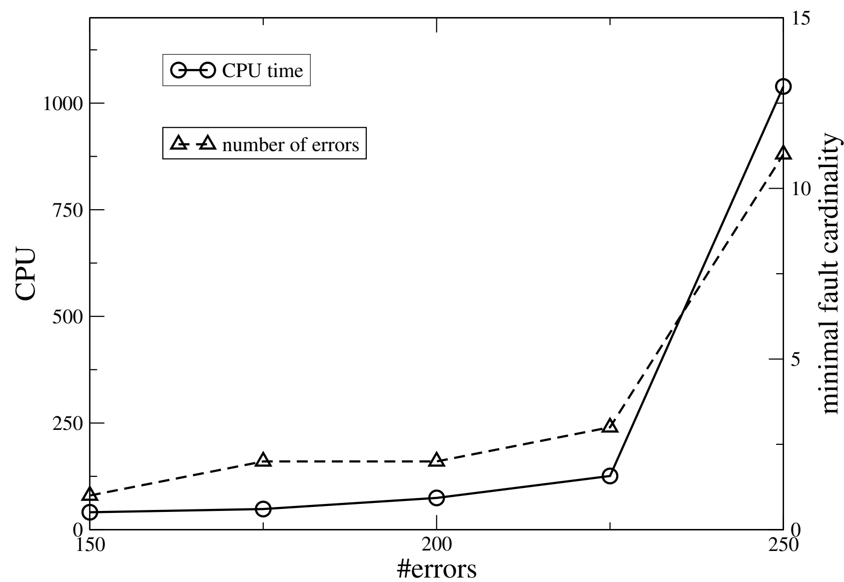

Figure 10. CPU time used by a PB-SAT solver to minimize the number of faulty cells to satisfy a constraint on the minimum number of errors at POs.

The CNF describing such a miter has been used to formulate the following PB problems: (1) determine the minimal number of faulty cells that result in $e_i = 1\ \forall i$ for $\eta_i = 32$ and (2) solve the same problem as in (1) but with constraints on the weights of neurons that have been randomly assigned. In case 1, the CPU time spent by the PB SAT solver/optimizer to provide the minimum ($A = 1$) is equal to 255.40 s. In case 2, the CPU time has been computed for 100 different sets of random weight values, resulting in a mean value equal to 220.87 s and a standard deviation equal to 68.32 s. In this latter case, the minimal value

of A ranges in the interval from 1 to 3 where the values larger than 1 are obtained only if several weights are at logic 0.

The results achieved using the PB optimizer show that, for modules of small size, testing and verification problems requiring optimization can also be approached.

5. Conclusions

Two new Boolean models for circuits in the presence of multiple faults are presented that can be used in SAT-based approaches for testing, diagnosis and reliability assessment in the presence of growing defect densities. They target cell faults resulting in monotonic changes of the cell function and faults affecting fan-out stems, respectively, thus covering a wide range of defects. This method has been compared to existing techniques from the point of view of the modeled faults as well as computational efficiency. In the case of defects resulting in monotonic faults, the proposed approach has an average (over ISCAS and ITC benchmarks) CPU time which is three times that of injecting output stuck-ats and one order of magnitude smaller than that of injecting input/output stuck-ats; however, these represent fewer general fault models. Moreover, by using a new kind of encoding for multiple-fault cardinality, we show that the costs related to the proposed approach can be made smaller than that of output stuck-ats. The method can be conveniently extended also to FPGAs featuring cells more complex than logic gates. In the case of faults affecting fan-out stems, the results show that the average number of solutions for a given primary output error configuration is equal to 73.1 while it is 105.0 when considering gate output stuck-ats, thus showing that, with almost the same CPU time, in fault diagnosis applications, we consistently reduce the number of possible culprits. Moreover, applications to problems requiring a PB formulation to track fault effects on numeric output values have also been investigated. In this last case, the application of the proposed approach to the improvement of design criticalities from the point of view of reliability remains a topic for further research.

Author Contributions: Conceptualization, M.F.; methodology, M.F. and M.D.; software, M.F. and F.D.; validation, F.D.; formal analysis, all the authors; data curation, F.D.; writing, M.F. and M.D.; visualization, F.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mishra, M.; Goldstein, S.C. Defect Tolerance at the End of the Roadmap. In *Nano, Quantum and Molecular Computing*; Springer: Boston, MA, USA, 2004; pp. 73–108.
2. Haselman, M.; Hauck, S. The Future of Integrated Circuits: A Survey of Nanoelectronics. *Proc. IEEE* **2010**, *98*, 11–38. [CrossRef]
3. Krishnaswamy, S.; Viamontes, G.F.; Markov, I.L.; Hayes, J.P. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In Proceedings of the Design, Automation and Test in Europe, Munich, Germany, 7–11 March 2005; pp. 282–287.
4. Biere, A.; Heule, M.; van Maaren, H. *Handbook of Satisfiability*; IOS Press: Amsterdam, The Netherlands, 2009; Volume 185.
5. Larrabee, T. Test pattern generation using Boolean satisfiability. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1992**, *11*, 4–15. [CrossRef]
6. Marques-Silva, J.; Sakallah, K. Boolean Satisfiability in Electronic Design Automation. In Proceedings of the 37th Annual Design Automation Conference, Los Angeles, CA, USA, 5–9 June 2000; pp. 675–680.
7. Smith, A.; Veneris, A.; Ali, M.F.; Viglas, A. Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2005**, *24*, 1606–1621. [CrossRef]
8. Feldman, A.; Pill, I.; Wotawa, F.; Matei, I.; de Kleer, J. Diagnosing Sequential Circuits as Boolean Satisfaction. In Proceedings of the 30th International Workshop on Principles of Diagnosis, DX 2019, Klagenfurt, Austria, 11–13 November 2019.
9. Fujita, M.; Mishchenko, A. Efficient SAT-based ATPG techniques for all multiple stuck-at faults. In Proceedings of the 2014 International Test Conference, Seattle, WA, USA, 20–23 October 2014; pp. 1–10.
10. Wang, P.; Moore, C.J.; Gharehbaghi, A.M.; Fujita, M. An ATPG method for double stuck-at faults by analyzing propagation paths of single faults. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *65*, 1063–1074. [CrossRef]
11. Riener, H.; Fey, G. Exact diagnosis using Boolean satisfiability. In Proceedings of the 35th International Conference on Computer-Aided Design, Austin, TX, USA, 7–10 November 2016; pp. 1–8.

12. Arumi, D.; Rodriguez-Montanes, R.; Figueras, J. Experimental characterization of CMOS interconnect open defects. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2007**, *27*, 123–136. [CrossRef]
13. Forero, F.; Galliere, J.M.; Renovell, M.; Champac, V. Analysis of short defects in FinFET based logic cells. In Proceedings of the 2017 18th IEEE Latin American Test Symposium (LATS), Bogota, Colombia, 13–15 March 2017; pp. 1–6.
14. Eén, N.; Sörensson, N. Translating pseudo-boolean constraints into SAT. *J. Satisf. Boolean Model. Comput.* **2006**, *2*, 1–26. [CrossRef]
15. Eén, N.; Sörensson, N. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 502–518.
16. Tseitin, G. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*; LOMI: Leningrad, Russia, 1968; pp. 115–125.
17. Panchapakeshan, P.; Narayanan, P.; Moritz, C.A. N 3 ASICs: Designing nanofabrics with fine-grained CMOS integration. In Proceedings of the 2011 IEEE/ACM International Symposium on Nanoscale Architectures, San Diego, CA, USA, 8–9 June 2011; pp. 196–202.
18. Amarú, L.; Gaillardon, P.E.; Mitra, S.; De Micheli, G. New logic synthesis as nanotechnology enabler. *Proc. IEEE* **2015**, *103*, 2168–2195. [CrossRef]
19. Brglez, F. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN. In Proceedings of the International Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, Kyoto, Japan, 5–7 June 1985; pp. 663–698.
20. Corno, F.; Reorda, M.S.; Squillero, G. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Des. Test Comput.* **2000**, *17*, 44–53. [CrossRef]
21. Sentovich, E.M.; Singh, K.J.; Lavagno, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; Stephan, P.R.; Brayton, R.K.; Sangiovanni-Vincentelli, A. *SIS: A System for Sequential Circuit Synthesis*; Technical Report No. UCB/ERL M92/41; EECS Department, University of California: Berkeley, CA, USA, 1992.
22. Mishchenko, A.; Chatterjee, S.; Brayton, R. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In Proceedings of the 2006 43rd ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 24–28 July 2006; pp. 532–535.