Journal of Electronic Testing (2024) 40:291–313
https://doi.org/10.1007/s10836-024-06119-5

Analysis of Combinational Circuit Failure Rate based on Graph Partitioning and Probabilistic Binomial Approach

Esther Goudet[1,2] · Fabio Sureau[2] · Paul Breuil[2] · Luis Peña Treviño[1,2] · Lirida Naviner[1] · Jean-Marc Daveau[2] · Philippe Roche[2]

Received: 8 August 2023 / Accepted: 18 April 2024 / Published online: 28 May 2024

Abstract
This paper studies the fault propagation and the correctness rate calculation of combinatorial circuits. We rely on circuit partitioning and on a probabilistic approach close to a binomial distribution, assuming some simultaneous faults have a certain probability to occur in the circuit's gates. We extend the results of our *Clusterized Probabilistic Binomial Reliability* model (CPBR), in which we obtained the results for several combinatorial multiplier designs, as seen in our previous publication. We now target non-arithmetic combinatorial netlists and, among them, a few circuits with flip-flop instances. We use the graph representation of the combinatorial netlists and we generalize our approach with a generic algorithm for CPBR. To develop this algorithm, we use some existing work on multilevel acyclic hypergraph partitioning, that we adapt to acyclic directed graphs. Furthermore, we address the problem of calculating correctness rates of circuits in cases where sequential flip-flops induce cycles in the graph. Our experiments show that our approach is capable of analysing the error and the correctness rates of significant non-arithmetic circuits, with an automatized and generic tool.

Keywords Logic function · Simultaneous faults simulation · Correctness rate · Acyclic graph partitioning · Signal probability reliability · Probability transfer matrix

1 Introduction

The down scaling of transistors induced by ultra-deep submicron technologies has been increasing digital sensitivity of digital systems to soft errors [30]. This is due to numerous physical factors such as increase in transistors density, smaller critical charges, junction leakages or aging of the components [1, 25, 27, 53]. In our study, we focus on the case where a soft error occurs in a combinatorial circuit but we assume that the electrical and the timing masking effect were ineffective. Thus, we aim at computing the error rate, or equivalently, the correctness rate of a combinatorial circuit as we only take into account the logical masking effect. Many methods exist to estimate this soft error rate, such as fault injection and radiation tests [2, 7, 31, 52]. ISO 26262 [22], the standard for safety systems in road motor vehicles, defines safety metrics for the safety assessment of digital systems such as *Single Point Failure Metrics* (*SPFM*), that results from *single point fault* and *residual faults*. A *single-point fault* is a fault that affect one element (logical gate) and corrupts the output of the affected module while a *residual fault* is a single-point fault that escape detection by a safety

Responsible Editor: L. M. Bolzani Pöhls

✉ Esther Goudet
esther.goudet@st.com

Fabio Sureau
fabio.sureau@st.com

Paul Breuil
paul.breuil@polytechnique.edu

Luis Peña Treviño
luishumberto.penatrevino@st.com

Lirida Naviner
lirida.naviner@telecom-paris.fr

Jean-Marc Daveau
jean-marc.daveau@st.com

Philippe Roche
philippe.roche@st.com

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

[2] Technology Design Platforms, STMicroelectronics, Crolles, France

mechanism(s). When targeting *ASIL C* (resp. *D*), *SPFM* must reach 97% (resp. 99%). Therefore, when evaluating such metrics, precision must be favored over execution time.

Some authors focus their research on localizing the most critical inputs, paths or gates in combinatorial circuits [38, 47–49]. However, the approach is not exhaustive and might overestimate the reliability of the circuits. Among the studies in this direction, the authors of [9] use a Monte Carlo simulation to estimate the error rate of the circuit, determined by logical masking. Then they assess the critical area of the circuit and evaluate the error rate accordingly. Other authors prefer exhaustive approaches to develop an accurate error rate calculation method. Among analytical and probabilistic models, many works use the *Probability Transfer Matrix (PTM)* approach [26, 33]. Nevertheless, the PTM time and space complexity explodes on large netlists, which makes it impractical on circuits used in current electronic designs. In the past years, many authors have tried to improve the PTM model [10–13, 23, 29, 35, 36, 39, 44, 50]. Cai and Chen [3] gives an approach that appears similar to PTM at first glance, but the authors claim it is linear depending on the number of gates of a circuit. In [36], the authors note a very interesting point on the accuracy of the PTM approach regarding the varying reliability values of the logic gates in combinatorial circuits. This work makes the PTM approach even more accurate and the comment on non-fixed values of the correctness rates of the gates is extensible to any other method. Jahanirad and Hosseini [24] also computes the probability of error of a gate depending on its CMOS transistor structure, which makes the estimation more precise. The authors of [41] develop an algorithm based on the Hadamard matrix product. They also focus on the physical dimension of the gate for the calculation of the output error rate of combinatorial circuits. Wang et al. [46] and Yu et al. [50] study the correlation coefficients to remedy the issue of the bad trade-off between time computation and accuracy when dealing with reconvergent signals. To tackle this issue of reconvergent paths, [51] states 3 types of correlated systems, from low correlated signals to strongly correlated ones, and uses a bitstream simulation technique to estimate the joint probabilities of signals. Chen et al. [6] studies the probability of error of a combinatorial circuit by calculating and comparing the probabilities of the faulted signals with the probabilities of the fault-free signals, while also taking into account the reliability of input signals. Stempkovskiy et al. [42] uses logic constraints on gates for a fast reliability analysis, but the algorithm is limited by the number of possibilities in gates with more than two input signals.

In our previous publication [14], we combined two exhaustive and probabilistic approaches to experiment a new probabilistic and partitioning approach that would reduce the complexity when processing significant circuits. The first model from which we drew our own *Clusterized Probability Reliability Model (CPBR)* is the *Hierarchical - Conditioned Probability Matrix (HCPM)* approach [15, 45]. This algorithm exploits the netlist topology and partitions the circuit in appropriate blocks to significantly decrease the computation time of the *Signal Probability Reliability Multi-Pass (SPRMP)* [12, 13] or PTM approaches. More recently, [34] also improved SPRMP, and as we mentioned above, a lot of authors have been working on signal correlations. Still, HCPM is interesting as it keeps the same accuracy as SPRMP while being capable of processing much bigger circuits. Although it is limited to netlists with ill-formed nested reconvergent paths, the progress made through partitioning are considerable, and the other authors of [5, 50] also had the idea to use partitioning to bind the complexity of their models. The second model from which we drew CPBR is the *Probabilistic Binomial Reliability (PBR)* model [8]. This method uses combinatorial enumeration and has a great accuracy in terms of error rates. Indeed, it takes into account the logical masking effect in the designs and it ignores the issues raised by reconvergent signals in HCPM. The original model, extensively detailed in [43], has a very high complexity and in 2020, the authors of [4] improved it to be capable of processing significant netlists with high precision. In [14], we tested our CPBR model on different sizes of a multiplier design. We compared the CPBR correctness rates with the PBR accurate method and they were quite close. Moreover, we studied how the error rates of the multiplier were evolving as we were changing the sizes of the clusters. This current paper extends upon the findings we presented in [14].

Previous approaches, such as the method described in our publications [14, 15], relied on manual cluster assignment, particularly suited for handling adders and multipliers. However, this method exhibits limitations in terms of performance and applicability to diverse circuit types. It is primarily effective for circuits consisting of a limited number of sub-modules arranged in regular sequences, such as arithmetic circuits.

In contrast, our current work aims to overcome these limitations by developing a generic algorithm applicable to any type of netlist. This algorithm is guided by the cluster criterion highlighted in [14] and informed by existing research on acyclic graph partitioning [37, 40]. A fundamental requirement of our approach is the necessity for the coarsened graph of clusterized combinatorial circuits to be acyclic. This acyclicity facilitates the propagation of error rates, or equivalently, correctness rates, from cluster to cluster and ultimately to the primary output signals. Within each cluster, the probability of error is computed using the PBR method and propagated using concepts such as the Ideal Transfer Matrix (ITM), PTM, and SPR matrix.

The paper is organized as follows. Section 2 outlines the main principles of PBR and presents our previous work on CPBR. Section 3 explains our generic partitioning algorithm

for CPBR. Section 4 describes how to compute the clusters logic functions. Also it shows how to simulate logic faults in the clusters with the error vectors used in the PBR method, thanks to the *Binary Decision Diagrams (BDD)*. Section 5 presents the complete process for treating non-arithmetic circuits with CPBR from a combinatorial point of view, and how to manage these circuits when sequential flip-flops are instantiated. Section 6 demonstrates the ability of our automatized algorithm to process significant non-arithmetical circuits. Eventually, Section 7 outlines the conclusions and on-going works.

2 Background

We are now doing a review of the two methods PBR and CPBR, and we provide a set of notations to refer to the concepts of these models. We also provide a calculation example to motivate the interest in evolving the PBR model towards the approximate CPBR method.

2.1 The PBR Model

The PBR model proposes an interesting fault model and it is exact. First, we explain the method in broad terms, then we justify the need to evolve this model to reduce its complexity.

2.1.1 Existing Work

The PBR model was developed in [8]. In this approach, the analysis of the correctness and error rates of a combinatorial circuit relies on a maximum number of simultaneous faulty logic gates in the netlist, called $k_{\max}$. The authors simulate all the possible locations for the simultaneous faults, from 1 fault to $k_{\max}$ faults in the circuit by unfolding some error vectors. The error vectors are simple binary vectors of length the number of gates in the targeted circuit. Each coordinate corresponds to a precise indexed gate and when the bit no. i is set to 1 in the error vector, the gate no. i in the circuit is considered as faulted. The error vectors are enumerated with the algorithm described in [28]. For each input vector of the netlist, the PBR algorithm compares the golden combinatorial result with the result that takes into account the propagation of the faults. We introduce the following notations to aid in the understanding of the PBR model.

- q is the probability for a faulty logic gate of the netlist to generate a correct output signal. Thus, $(1-q)$ is the probability for a faulty logic gate to give a wrong output signal.
- N_G is the number of logic gates in the circuit.
- N is the number of input signals of the circuit.
- $\mathcal{S}(.,.)$ is the logic function standing for a primary output $\mathcal{S}$ of the circuit. The function takes as argument an input vector $\mathcal{I}$ and an error vector $\mathcal{E}$.
- $\mathcal{E}_k^j$ is an error vector of bitsize N_G. Each bit of $\mathcal{E}_k^j$ stands for a gate in the circuit and $k \geq 1$ stands for the number of simultaneous faulty logic gates in the netlist. $j \in \left[\!\left[1, \binom{N_G}{k}\right]\!\right]$ is the number of the configuration of k bits set to 1 among the N_G coordinates of $\mathcal{E}_k^j$.
- $p_{\text{err}_\mathcal{S}}$ is the probability for a primary output $\mathcal{S}$, to be faulty.

Let $\mathcal{S}$ be a primary output of a combinatorial circuit, affected by N primary input signals. At the end of the PBR algorithm, we weighed all the comparisons between the correct value of $\mathcal{S}$ and the erroneous one, for each pair of an input vector and an error vector. Suppose there are at most $k_{\max}$ faults in the circuit with a probability $(1-q)$ for each of them to occur. The probability for $\mathcal{S}$ to be bit-flipped is then:

$$p_{\text{err}_\mathcal{S}} = \sum_{k=1}^{k_{\max}} \left[\sum_{j=1}^{\binom{N_G}{k}} \sum_{z=0}^{2^N-1} \mathcal{S}(z,0) \oplus \mathcal{S}(z, \mathcal{E}_k^j) \right] \times \frac{(1-q)^k q^{N_G-k}}{2^N \times \sum_{k=1}^{k_{\max}} \binom{N_G}{k}} \tag{1}$$

with $\oplus$ denoting the XOR operation between two boolean variables. The formula in (1) is also reported and explained in [8] and [4].

2.1.2 Generalisation of the Probability of Error for a Set of Signals

In [14] we generalized (1) to the probability for a set of several dependant signals to be in a certain state. Let $\{S_1, \ldots, S_n\}$ be a set of n dependant signals. We note $\widetilde{S_n}$ the n-tuple $(S_1, \ldots, S_n)$. Let $\widetilde{X_n} = (X_1, \ldots, X_n)$ also be a n-tuple in $\{0,1\}^n$. If we take the same notations as in the previous paragraph, then the probability $p_\mathcal{I}(\widetilde{X_n}, q)$ that $\widetilde{S_n} = \widetilde{X_n}$, for a certain input vector $\mathcal{I}$ of the circuit, is calculated in as follows:

$$p_\mathcal{I}(\widetilde{X_n}, q) = \frac{\sum_{k=1}^{k_{\max}} \left[(1-q)^k \times q^{N_G-k}\right] \sum_{l=1}^{C_k^{N_G}} C_{\widetilde{S_n}, \mathcal{E}_k^l, \mathcal{I}, \widetilde{X_n}}}{\sum_{k=1}^{k_{\max}} C_k^{N_G}}$$

$$\text{with} \quad \begin{cases} C_k^{N_G} = \binom{N_G}{k} \\ C_{S_i, \mathcal{E}_k^l, \mathcal{I}, X_i} = S_i(\mathcal{I}, \mathcal{E}_k^l) \oplus X_i \\ C_{\widetilde{S_n}, \mathcal{E}_k^l, \mathcal{I}, \widetilde{X_n}} = \begin{cases} 1 & \text{if } \exists i \in [\![1,n]\!] : C_{S_i, \mathcal{E}_k^l, \mathcal{I}, X_i} = 1 \\ 0 & \text{if } \forall i \in [\![1,n]\!] \quad C_{S_i, \mathcal{E}_k^l, \mathcal{I}, X_i} = 0 \end{cases} \end{cases} \tag{2}$$

2.1.3 Complexity Motivation for the CPBR Model

In [14], we worked on the *Clusterized Probability Binomial Reliability (CPBR)* model, which improves the PBR model. We motivated the partitioning approach to reduce the complexity of the PBR model, which strongly depends on the number of gates and inputs of a circuit. Indeed, with the parameters introduced in the paragraph 2.1.1, the time complexity of the PBR approach is:

$$\mathcal{T}_{\text{PBR}} = \mathcal{O}\left(2^N \times \sum_{k=1}^{k_{\max}} \binom{N_G}{k}\right) \tag{3}$$

And an inferior bound on the PBR approach memory complexity is:

$$\mathcal{I}nf\ \mathcal{S}_{\text{PBR}} = \mathcal{O}\left(N_G \times \sum_{k=1}^{k_{\max}} \binom{N_G}{k}\right) \tag{4}$$

2.2 The CPBR Model

The CPBR model approximates the PBR model by using circuit partitioning. The goal of this method is to obtain error rates very close to those produced by the PBR method, while reducing computation costs. As the processing of a partitioned circuit is quite complex, we provide a calculation example with the CPBR method. Then, we justify the need to evolve the implementation of the model, which so far has been tested only "by hand", in a rather experimental way.

2.2.1 Previous Work

Goudet et al. [14] explains how to propagate the error and the correctness rates of the signals of a combinatorial circuit after its partitioning, with the help of the *Probability Transfer Matrix (PTM)* [33], the *Ideal Transfer Matrix (ITM)*, and the *Signal Probability Reliability (SPR)* [12] matrices. We detailed the calculations of the ITM and the PTM matrices of the outputs of a cluster, either for a single signal or for a group of several signals. Figure 1 shows how to propagate the probabilities of the signals from cluster to cluster to the primary outputs. It also outlines the issue raised by the reconvergent signals. Those signals cannot always be gathered in one same cluster, thus, they induce wrong calculations when they reconverge towards one cluster through the partition chain. However, as we show in [14] on several sizes of a multiplier design, the CPBR model results have a very low discrepancy with the PBR model rates, even though it is not always able to gather a reconvergent path in a single cluster. Moreover, in comparison with the PBR approach, the CPBR model never overestimates the correctness rates of the output signals.

2.2.2 CPBR Example

We now show an example of the treatment of a clusterized circuit with CPBR. This example circuit is shown in Fig. 1. To obtain the SPR matrix of a group of one or several output signals of a cluster, we first determine the SPR input matrix of the cluster. This gives the joint probabilities of the input signals to be on the 4 states 1_c (1 correct), 0_c (0 correct), 1_i (1 incorrect) and 0_i (0 incorrect). For example, the $\text{SPR}_{\text{input}}$ matrix of the cluster C_3 in Fig. 1 is calculated below:

$$\text{SPR}_{\text{input}_{C_3}} = \text{SPR}_{S_2} \otimes \text{SPR}_{S_3} \otimes \text{SPR}_{S_4}, \tag{5}$$

where $\otimes$ denotes the Kronecker product between two matrices.

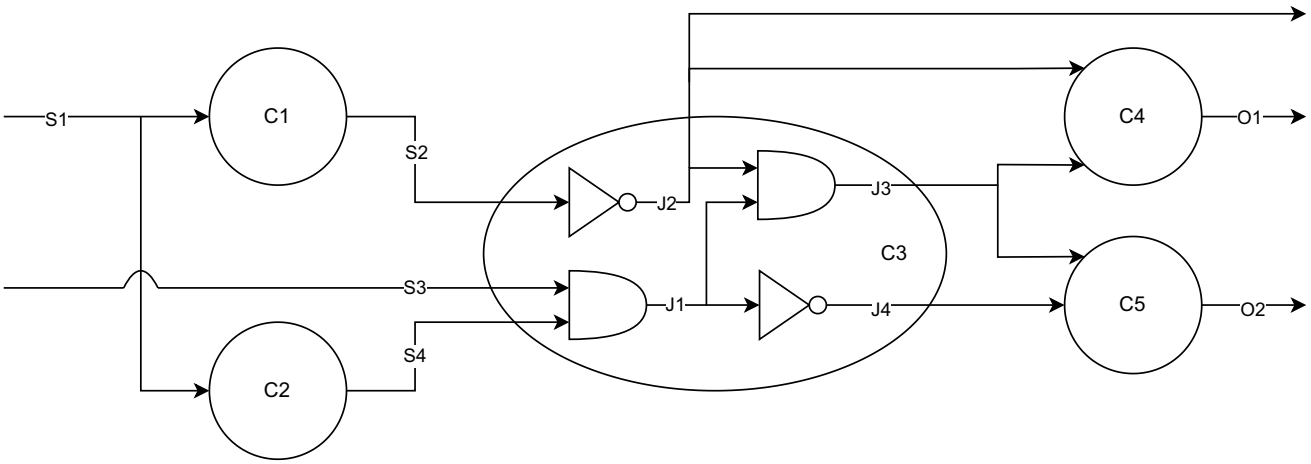


Fig. 1 Example of a clusterized circuit to be processed with the CPBR approach

Next, given a group of output signals, we compute the ITM of the signals, which is basically their truth table. After that, given some probabilities for the signals to be bit-flipped, we compute the PTM of the set of signals. The PTM of a set of several signals stores the joint probabilities for the signals to be on all possible combinations of states, given the errors in some gates of the cluster or the circuit. The probabilities in the PTM are obtained with (2), as explained in [14]. Lastly, we compute the following matrix product to obtain the joint probabilities of the output signals to be on the 4 states $1_c, 1_i, 0_c, 0_i$:

$$\mathrm{SPR}_{\mathrm{output}} = {}^t\mathrm{ITM} \times \mathrm{SPR}_{\mathrm{input}} \times \mathrm{PTM}$$

The correctness rate of the output set is then:

$$\mathbb{P}(\mathrm{output} \in \{0_c, 1_c\}) = \mathrm{Tr}\,(\mathrm{SPR}_{\mathrm{output}})$$

Note that for the sake of simplicity, we always consider that the group of signals of a cluster for which we compute a set of an ITM, a PTM and a SPR matrix, takes as inputs all the input signals of the cluster. This is easier since we then only need to compute one $\mathrm{SPR}_{\mathrm{input}}$ matrix for all the calculations on all the output signals of the cluster. To provide a more concrete understanding, we will next demonstrate the treatment of the cluster C_3 in Fig. 1 with the CPBR method.

The internal processing of C_3 with CPBR is the following. We determine a $k_{\max}$ parameter dedicated to C_3, which is comprised between 1 and 4, the number of logic gates in C_3. C_3 has 3 input signals: S_2, S_3 and S_4.

- J_2 is a primary output of the circuit so we compute a SPR output matrix only for it. In terms of inputs, we calculate the ITM and the PTM of J_2 as if it were affected by the 3 input signals, S_2, S_3 and S_4. The truth table and the ITM of J_2 are:

$$\begin{array}{cccc} S_2 & S_3 & S_4 & J_2 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{array} \Longrightarrow \mathrm{ITM}_{J_2} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

  We apply (1) to compute the probability that J_2 is bit-flipped, with the following parameters:

  - N, the number of input signals of C_3,
  - N_G, the number of gates in C_3,
  - $k_{\max}$, the maximum number of simultaneous faults in C_3,
  - q, the probability for a faulty gate to give a correct output signal.

  We call $p_{\mathrm{err}_{J_2}}$ the result of (1), with the aforementioned parameters. The PTM of J_2 is then:

$$\mathrm{PTM}_{J_2} = \begin{pmatrix} 1-p_{\mathrm{err}_{J_2}} & p_{\mathrm{err}_{J_2}} \\ 1-p_{\mathrm{err}_{J_2}} & p_{\mathrm{err}_{J_2}} \\ 1-p_{\mathrm{err}_{J_2}} & p_{\mathrm{err}_{J_2}} \\ 1-p_{\mathrm{err}_{J_2}} & p_{\mathrm{err}_{J_2}} \\ p_{\mathrm{err}_{J_2}} & 1-p_{\mathrm{err}_{J_2}} \\ p_{\mathrm{err}_{J_2}} & 1-p_{\mathrm{err}_{J_2}} \\ p_{\mathrm{err}_{J_2}} & 1-p_{\mathrm{err}_{J_2}} \\ p_{\mathrm{err}_{J_2}} & 1-p_{\mathrm{err}_{J_2}} \end{pmatrix}$$

  Lastly, the SPR matrix of J_2 is obtained with the following calculation:

$$\mathrm{SPR}_{J_2} = {}^t\mathrm{ITM}_{J_2} \times \mathrm{SPR}_{\mathrm{input}_{C_3}} \times \mathrm{PTM}_{J_2}$$

- J_2 and J_3 are directed to the same downstream cluster after C_3 so we compute their joint SPR output matrix. The truth table and the ITM of (J_2, J_3) are:

$$\begin{array}{ccccc} S_2 & S_3 & S_4 & J_2 & J_3 \\ \hline 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{array} \Longrightarrow \mathrm{ITM}_{(J_2,J_3)} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

  Next, we build the PTM. For each input vector $\mathcal{I} \in \{0,1\}^3$ of C_3 and for each couple $\widetilde{X_{23}} = (X_2, X_3) \in \{0,1\}^2$, we compute the probability $p_{\mathcal{I}}(\widetilde{X_{23}}) = \mathbb{P}(\widetilde{J_{23}} = \widetilde{X_{23}} | \mathcal{I})$, where $\widetilde{J_{23}} = (J_2, J_3)$, with the following equation:

$$\forall \mathcal{I} \in \{0,1\}^3, \forall \widetilde{X_{23}} \in \{0,1\}^2,$$

$$p_{\mathcal{I}}(\widetilde{X_{23}}, q) = \frac{\sum_{k=1}^{k_{\max}} \left[(1-q)^k \times q^{4-k}\right] \sum_{l=1}^{C_k^4} C_{\widetilde{J_{23}}, \mathcal{E}_k^l, \mathcal{I}, \widetilde{X_{23}}}}{\sum_{k=1}^{k_{\max}} C_k^4} \tag{6}$$

  $p_{\mathcal{I}}(\widetilde{X_{23}}, q)$ is the probability that $(J_2, J_3) = (X_2, X_3)$ given the above parameters. Note that (6) is just the simplified version of (2), for only two variables. $p_{\mathcal{I}}(\widetilde{X_{23}}, q)$ is inserted in $\mathrm{PTM}_{(J_2,J_3)}$ at the place of the corresponding coefficient, as below:

$$\mathrm{PTM}_{(J_2,J_3)} = \begin{pmatrix} p_{000}(00) & p_{000}(01) & p_{000}(10) & p_{000}(11) \\ p_{001}(00) & p_{001}(01) & p_{001}(10) & p_{001}(11) \\ \vdots & \vdots & \vdots & \vdots \\ p_{111}(00) & p_{111}(01) & p_{111}(10) & p_{111}(11) \end{pmatrix}$$

Once we have $\mathrm{ITM}_{(J_2,J_3)}$ and $\mathrm{PTM}_{(J_2,J_3)}$, we deduce $\mathrm{SPR}_{(J_2,J_3)}$:

$$\mathrm{SPR}_{(J_2,J_3)} = {}^t\mathrm{ITM}_{(J_2,J_3)} \times \mathrm{SPR}_{\mathrm{input}_{C_3}} \times \mathrm{PTM}_{(J_2,J_3)}$$

$\mathrm{SPR}_{(J_2,J_3)}$ is then the SPR input matrix of the cluster C_4.

- The calculation of the joint SPR matrix for (J_3, J_4), which is needed for the SPR input matrix of the cluster C_5, follows exactly the same principle as $\mathrm{SPR}_{(J_2,J_3)}$. Thus, we do not detail it further.

Note that the ITMs we give for J_2 and for the couple (J_2, J_3) are only valid if we fixed a certain order for unfolding the inputs of the cluster. In the CPBR approach, each time we apply the PBR algorithm to a cluster, we have to pay attention to unfold the inputs of the cluster in the exact same order, whether we compute its SPR input matrix or we compute the ITM and the PTM of one of its group of output signals. Indeed, the order we used here for the ITMs and PTMs of J_2 and (J_2, J_3) is precisely the same as the one used in (5). This consistency in the order of input variables is crucial when processing a circuit or cluster to ensure consistent and accurate results.

2.2.3 Extension and Improvement of the Previous Work

In [14], we gave some results on significant arithmetic circuits, like a 16×16 multiplier. We studied the differences in results between the PBR and the CPBR approaches on smaller sizes of the multiplier. We also explored the effects of the sizes and the shapes of the clusters on the correctness rates of the circuit's output signals. Nevertheless, all the results in [14] were obtained with handmade clusters, which were easy to draw with the visual structure of the multiplier. We now aim at giving some results on a larger slice of non-arithmetic circuits. To demonstrate our progress on the CPBR model, we introduce a new automatized and generic algorithm, capable of designing appropriate partitions of the circuits of the benchmarks we studied. After its partitioning phase, the program computes the ITM and the PTM of the clusters. Next, it propagates the SPR matrices through the clusters chain. The correctness rates we present in that article are calculated with this algorithm.

3 A Generic Partitioning Algorithm for the Circuits Graphs

In this section, we contextualize the partitioning algorithm within the CPBR method. We detail the implementation we have chosen, revisiting the references that allowed us to reprogram an efficient acyclic partitioning algorithm, which met exactly the criteria we were looking for.

3.1 Motivation for a Generic Partitioning Algorithm

The first step in a generic CPBR algorithm capable of processing any type of circuit, is a generic partitioning algorithm. The study of the multipliers outlined several criteria for an ideal partition for CPBR, such as the acyclicity of the coarsed graph after the circuit is clusterized and the scalability of the clusters for the PBR model. Moreover, we hope to achieve a partitioning algorithm that minimizes the spread of the targets of one signal in several clusters. The more we consider the correlation of the signals, the more accurate our correctness rate calculations will be. For example on the 8×8 and 16×16 multipliers, we manually designed a partition such that every cluster is not affected by more than 2 or 3 upstream clusters, and any output signal of a cluster targets only one other downstream cluster. Thus, in terms of the aforementioned criteria, a generic clustering algorithm should not give a worse partition than the one in [14].

In fact, many applications in the digital circuit design field require an acyclic partitioning algorithm that both allows to control the size of the clusters and to generate clusters with low correlation, based on the signals of the studied circuit. Considering the shapes of the circuits we focus on, we have searched for an acyclic partitioning algorithm for directed acyclic graphs, that minimizes the cut-net between the clusters. The cut-net of a graph is defined as the number of edges that, after partitioning, affect several clusters. The objective of minimizing the cut-net of a graph when partitioning is therefore synonymous of minimizing the spread of reconvergent paths across the clusters. Actually this type of algorithm is the subject of active research in the industrial and Applied Mathematics field. The partitioning algorithm we use relies on the approaches of [37, 40]. These two works have the advantages of presenting a partitioning method that is fast, acyclic and that minimizes the cut-net of a graph. The process explained in [37] consists of three phases, but we only use the two last ones, namely the acyclic coarsening recursion and the refinement process on the acyclic coarsened sets.

3.2 The Combinatorial Circuit Verilog Netlist Parsing

The tool we built uses the Verilog netlists of the combinatorial circuits. The first task to process a circuit is to parse its Verilog gate netlist into a C++ graph. To do so, we used the open-source parser Icarus Verilog [18], which translates the logic elements of the circuit, such as the signals and the logic gates, in C++ structures. At first glance, the GitHub project [18] shows that Icarus is intended to be used as a simulator or as a logic synthesizer, but we limit our use to the parsing functionality to traverse netlists. Although Icarus flattens the hierarchy of the Verilog modules in its C++ structures, it is fortunately possible to use some subtleties like the indices of the objects, to reconstruct the initial information. Also, it provides a lot of features that allow to identify the properties of the logic and sequential items in the Verilog netlists, such as their names, types, connections, scopes, and so on. So we browsed the internal structure of Icarus to make our own API. We focused only on the resulting combinatorial structures, that we incorporated in our object oriented back-end. Once we completed the back-end API, we carry out some processing to reproduce a graph structure faithful to the circuit, in which all the logic source and target nodes are correctly connected. This graph is built with the C++ *Boost Graph Library (BGL)* [21]. The logic gates and the flip-flops instances of the netlists are transformed in vertices of the graph, while the Verilog components like wires, arrays and buses are transformed in edges. The graph class we use to translate the circuits is an adjacency list, because of the sparsity of the graph representation of the netlists. Once that is done, we can manipulate the circuit structure in a more convenient way and to further clusterize it.

3.3 The Partitioning Algorithm

Definitions and Notations First, we briefly introduce some notations to make the explanations about the partitioning algorithm clearer.

- A graph G = (V, E) is defined by a set V of n vertices and a set E of m edges between nodes in V. If $e \in$ E is an edge from v_1 to v_2, with $(v_1, v_2) \in \mathrm{V}^2$, we also note $e = (v_1, v_2)$.
- The vertices and edges weights are defined by the function w:

$$w : V \cup E \rightarrow \mathbb{R}^+ \tag{7}$$

w is extended to any subset of V or E in the following way:

$$\begin{aligned} &\forall U \subset V,\ w(U) = \sum_{v \in U} w(v) \\ &\forall F \subset E,\ w(F) = \sum_{e \in F} w(e) \end{aligned} \tag{8}$$

- The indegree of a vertex $v \in$ G is the number of edges that target v. The top level of v, noted `top[v]`, is the length of the longest path from any vertex with indegree zero, to v.
- Let $\mathcal{C}$ be a set of clusters of G after we partitioned the graph. We note $\mathrm{G}_{|\mathcal{C}} = (\mathrm{V}_{|\mathcal{C}}, \mathrm{E}_{|\mathcal{C}})$ the coarsened graph of the clusterized circuit. $\mathrm{G}_{|\mathcal{C}}$ is a graph with one node per cluster and one edge for each pair of connected clusters. The weight of a vertex $v_{\mathcal{C}} \in \mathrm{V}_{|\mathcal{C}}$ is the sum of the weights of the vertices encompassed by $v_{\mathcal{C}}$ in V. The weight of an edge $(V_1, V_2) \in \mathrm{E}_{|\mathcal{C}}$ is the sum of the weights of all the distinct edges in E that connect a pair of two elementary nodes $(v_1, v_2) \in \mathrm{V}^2$, with v_i encompassed by V_i for $i \in \{1, 2\}$.
- In the graph of a combinatorial netlist, before the partitioning, every vertex $v \in$ V stands for a logic node of the circuit, every $e \in$ E stands for a signal between two logic nodes in the circuit, and $w(v) = w(e) = 1$.

CPBR Constraints for a Combinatorial Circuit Partition At the beginning of the partitioning algorithm for the combinatorial circuits graphs, we set a bound `NI` on the number of input nets that enter a cluster and a bound `NG` on the number of logic nodes inside a cluster. This is done so that the clusters are small enough for their PBR analysis to converge quickly, since the time complexity of the PBR approach is exponential accordingly to the number of inputs signals, while both its time and memory complexities heavily depend on the number of gates in a circuit. Nevertheless, if `NI` and `NG` get too small, the gap between the results of the PBR and the CPBR methods will widen because the correctness rates of the second will become pessimistic. Indeed, the smaller the clusters, the less effective logical masking is.

Acyclic Coarsening The current step makes a first acyclic partition of a circuit graph. In 2019, the authors of [16] stated a theorem based on top level criteria for an acyclic partition of a graph. In 2020, the authors of [37] deduced their own theorem from the one of [16], to build an acyclic partition of a hypergraph. The theorem in [37] relies on two conditions that are close to the ones of the theorem in [16], and the resulting algorithm to obtain the acyclicity of the partition of a hypergraph is widely detailed in [37]. Moreover, it is easily adaptable to an acyclic directed graph, which includes all circuit graphs we target. The success of the recursive process described in [37] relies on the fact that at any round no. n of the recursion, if the initial partition P_n is acyclic and fulfills the Condition 1, it builds an acyclic partition P_{n+1}, derived from P_n, that still fulfills the Condition 1.

Condition 1 $C = (C_1, \ldots, C_k)$ is a partition of a directed acyclic graph G=(V,E) such that

$$\forall i \in [\![1, k]\!],\ \exists t_i \in \mathbb{N}\ /\ \forall u \in C_k \cap V,\ \texttt{top[u]} \in \{t_i, t_i + 1\}.$$

The acyclic coarsening algorithm on a directed acyclic graph, adapted to our clusters sizes constraints NI and NG, is the following. At first, every vertex stands for a singleton cluster. For each cluster composed of only one node, we rate each one of its neighbors using the heavy-edge rating function of the equation (4.3) in [40]. The authors of [40] were working on a hypergraph but we can use the simplified version of the equation for a graph G = (V, E), as expressed below:

$$\forall (u, v) \in \mathrm{V} \times \mathrm{V},\ r(u, v) := \frac{w((u, v))}{w(u) \times w(v)}$$

Now, let us call u the node in a singleton cluster. We select the highest-rated neighbor v of u. We note t the smallest top level of the nodes in v's cluster. If top[u] $\in \{t, t + 1\}$, and if the move does not imply more than NI input signals and more than NG logic nodes in v's cluster, we temporarily add u to it. Next we check that no cycle is induced by this move in the coarsened graph. To do so, we reproduce the exact verification method explained in the paragraph *3.2 Acyclic Coarsening* of [37]. If the algorithm finds a cycle, u is removed from v's cluster, otherwise it is left inside. We repeat this step for all nodes that are alone in their cluster. Once we have finished processing all the singleton clusters, we contract all the nodes of each new cluster inside one new node. We start the next round of the current step on the resulting coarsened graph. We repeat this operation on the newly obtained graph until all of its clusters after the execution of a coarsening step are still singletons. In [37], the authors use top levels and reversed top levels, i.e., the maximum distance from a node to any node with outdegree zero, each taking turns every other time. This precaution is important since it decreases the risks of building an unbalanced partition. In the worst case when the clusters have poorly distributed weights, a terminal node of the circuit graph remains in a singleton cluster in the final partition. This really skews downward the correctness rate of the corresponding output signal of the circuit.

Acyclic Uncoarsening Refinement This second phase is described in the paragraph *3.3 Acyclic Refinement* of [37]. It uses the *Algorithm Outlines* section of the paragraph *4.4 Localized FM Local Search* in the article [40], which itself uses the delta-gain updates computations of the algorithm of Papa and Markov [32]. During that phase of the algorithm, when moving a node, we still ensure that the move does not result in a number of input signals greater than NI and in a number of nodes greater than NG for the enlarged cluster. If the move violates this condition, we do not consider it. In contrast to the work of [40], we did not implement the net locking and caching techniques detailed in the *Algorithm Outlines* section, that would allow for further optimization of the execution time.

4 Generic Computation of Logic Functions

In the few studies we have conducted on the multiplier in [14], the sequence of operations on the clusters was manually ordered, and we had recoded the logical functions of the clusters we had designed ourselves. This was acceptable for the experimentation of our method on circuits with very repetitive structures, but it is obviously not viable for generalizing and automating our CPBR method, which we need to scale up on non-arithmetic circuits from classic benchmarks. Therefore, we have found a way to automate these two operations in our C++ tool.

4.1 Clusters Topological Ordering

After partitioning a circuit graph with the algorithm described in Section 3, we first determine how the clusters are linked to one another. More precisely, the clusters are indexed, and, to each cluster C, we attach a table in which we map the non primary output signals of C to the cluster or clusters that it targets. This way, for each cluster, we precisely know its input signals, their number and whether they are some primary inputs of the circuit or if they come from upstream clusters. This is very helpful to determine the ITM and PTM to be computed for a cluster, and after that, to chain the SPR matrices for the probability propagation. For example, the input edges map between the clusters in Fig. 2 is given in Table 1. For the cluster no. 4, we would for instance compute an ITM and PTM couple for each edge in $\{J \to L, J \to O, H \to M\}$.

Since the partitioning algorithm gives an acyclic coarsened graph, we are able to build a topological sort on the clusters. In fact, the mapping between the clusters' output signals and their targets makes obtaining this topological sort convenient. More precisely, we obtain the topological sort on the clusters with the algorithm 1, in which the condition at line 12 is checked by the mapping performed as explained above. For example, a topological sort on the clusters in Fig. 2 is:

$$C1 \prec C2 \prec C3 \prec C4 \prec C5 \prec C8 \prec C7 \prec C6 \prec C9 \prec C10$$

Algorithm 1 Clusters topological sort

```
Input: g           :  the circuit graph
       ClusterList :  the list of all clusters
vec_post = ∅;
vec_in = ClusterList;
while vec_post.size() < ClusterList.size() do
    for C ∈ vec_in do
        C_inputs = ∅;
        C_primary_inputs = ∅;
        for C_input ∈ C.Input_Edges do
            src = source(C_input, g);
            if src ∉ C_inputs then
                C_inputs.push_back(src);
            end
            if ∀J ∈ vec_in \ {C}, src ∉ J.Vertices then
                if src ∉ C_primary_inputs then
                    C_primary_inputs.push_back(src);
                end
            end
        end
        if C_inputs == C_primary_inputs then
            vec_post.push_back(C);
            vec_in.erase(C);
            break;
        end
    end
end
return vec_post;
```

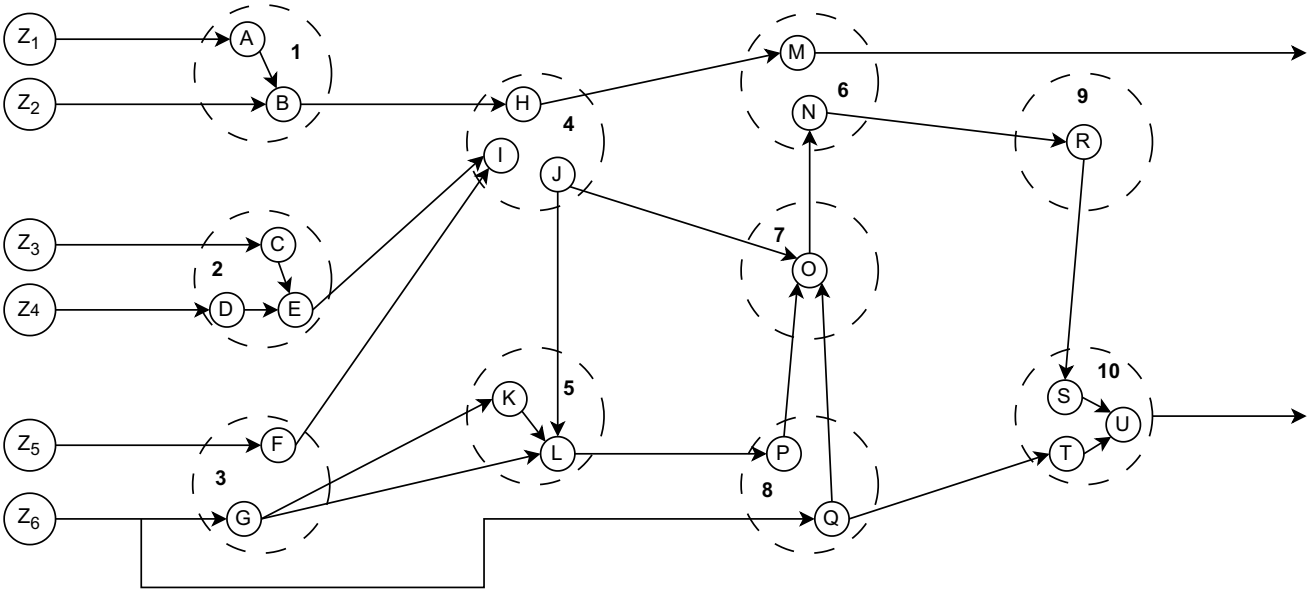


Fig. 2 Example of a clusterized circuit coarsened graph

Table 1 Input edges mapping for the clusters in Fig. 2

Source Cluster	Attached Map Table	
	Key: Edge	Value: Target Cluster
1	B → H	4
2	E → I	4
3	F → I	4
	G → K	5
	G → L	
4	H → M	6
	J → L	5
	J → O	7
5	L → P	8
6	N → R	9
7	O → N	6
8	P → O	7
	Q → O	
	Q → T	10
9	R → S	10

This topological sort of the clusters is very useful for a point we already stressed in the paragraph 2.2.2. Suppose we want to compute the SPR matrix of a certain group O of output signals of a cluster C. To do so we compute a matrix multiplication like ${}^t\text{ITM}_O \times \text{SPR}_{\text{input}_C} \times \text{PTM}_O$. In that calculation, it is important that each row no. i of ITM_O and of PTM_O correspond to the same input vector of the cluster C. For example if C has 3 input signals i_1, i_2, i_3 and the row no. 0 of ITM_O corresponds to the input vector $(i_1, i_2, i_3) = (0, 0, 0)$, then it must be the same for the row no. 0 of PTM_O. In addition, the Kronecker product intended to compute $\text{SPR}_{\text{input}_C}$ cannot be done in a random order on the input signals of C. It must be computed in the same order as the one used for listing the input vectors when computing the ITM and the PTM matrices. For instance, ITM_O and PTM_O beeing computed with the unfolding below implies the following calculation for $\text{SPR}_{\text{input}_C}$:

$$[i_1, i_2, i_3] = \left\{ \begin{array}{ll} 1^{\text{st}} & : [0,0,0] \\ 2^{\text{nd}} & : [0,0,1] \\ 3^{\text{rd}} & : [0,1,0] \\ 4^{\text{st}} & : [0,1,1] \\ 5^{\text{st}} & : [1,0,0] \\ 6^{\text{st}} & : [1,0,1] \\ 7^{\text{st}} & : [1,1,0] \\ 8^{\text{st}} & : [1,1,1] \end{array} \right\}$$

$$\Longrightarrow \text{SPR}_{\text{input}_C} = \text{SPR}_{i_1} \otimes \text{SPR}_{i_2} \otimes \text{SPR}_{i_3}$$

The topological sort of the clusters, combined with the topological sorts of the vertices inside the clusters, secures a constant order to unfold the input vectors of all clusters, when computing their SPR input matrices and the ITMs and PTMs of their outputs.

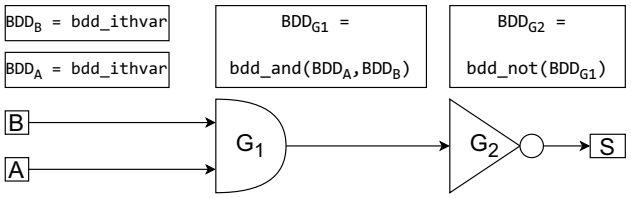


Fig. 3 Example of a BDD operation on a little sequence

4.2 Clusters Logic Functions Assessment with BDD

The assessment of the logic functions of the clusters is done with the BuDDy tool [17], which implements the *Binary Decision Diagrams (BDD)* in C++. We attach one `bdd_ithvar`, meaning one positive assertion, to each input signal node of the cluster. For any logic gate in the cluster, we attach a BDD item to its logic node in the cluster graph, depending on its logic function (`bdd_and`, `bdd_not`, `bdd_buf`, `bdd_or`, `bdd_nor`, ⋯). A BDD item, or equivalently a BDD function, can be composed with other similar objects. This allows us to link BDD nodes between themselves and, eventually, to obtain the logic function of a cluster through a BDD. For instance in Fig. 3, the source of the BDD item of the NOT node is the BDD object of the AND node, which itself has for sources the BDDs of the signals A and B.

To simulate a fault on a logic gate $\mathcal{L}$ in a cluster C, we compose the BDD item corresponding to the occurrence of the fault, noted $\text{BDD}_{\mathcal{F}\mathcal{L}}$, with the BDD of $\mathcal{L}$, noted $\text{BDD}_{\mathcal{L}}$. We call $\text{BDD}_{\mathcal{E}\mathcal{L}}$ the resulting BDD, that corresponds to the output signal from $\mathcal{L}$. When the fault on $\mathcal{L}$ is disabled, $\text{BDD}_{\mathcal{F}\mathcal{L}} = \texttt{bdd\_nithvar}$, meaning a negative assertion, thus, $\text{BDD}_{\mathcal{E}\mathcal{L}} = \texttt{bdd\_xor}(\texttt{bdd\_nithvar}, \text{BDD}_{\mathcal{L}}) = \text{BDD}_{\mathcal{L}}$. On the contrary, when the bit of $\mathcal{L}$ is raised to 1 in the error vector, we have $\text{BDD}_{\mathcal{E}\mathcal{L}} = \texttt{bdd\_xor}(\texttt{bdd\_ithvar}, \text{BDD}_{\mathcal{L}}) = \texttt{bdd\_not}(\text{BDD}_{\mathcal{L}})$.

5 CPBR Algorithm on Non-arithmetic Circuits: Methods to Process ISCAS85 and ISCAS89 Benchmarks

The ISCAS85 benchmark, composed of combinational netlists, was first introduced at the International Symposium of Circuits and Systems in 1985. The ISCAS89 benchmark, composed of sequential circuits, is an extension of the ISCAS85 benchmark and was introduced in 1989. Succeeding in making our CPBR method work on these two benchmarks is the first step in generalizing our model to any type of circuit.

5.1 CPBR Algorithm for ISCAS85 Benchmark

We used the Verilog ISCAS85 gate netlists of the webpage [19]. The circuits of this benchmark are very simple gate netlists, with no back-edge and no latch, flip-flop or

sub-scope. For each circuit in the ISCAS85, we applied our parsing API using IcarusVerilog to build the C++ circuit graph as explained in the paragraph 3.2. More precisely, we draw one logic node per logic gate in the circuit and one artificial "true" node per primary input signal of the verilog netlist. Next we draw one edge per signal between two logic nodes in the circuit. Also, if a logic node $\mathcal{L}$ has a primary input signal $\mathcal{I}$ as input, we draw an edge in the graph between the "true" node for $\mathcal{I}$ and the logic node for $\mathcal{L}$. We call the circuit graph `g`.

To be more efficient and more accurate, we compute the CPBR algorithm on the cones of influence of the primary outputs. We loop over all the primary output signals of the circuit. For any of them, we consider `PO` the source node of the signal. We extract the graph of the cone of influence of `PO` from `g`. We call the extracted graph `PO_subgraph`. We partition `PO_subgraph` with the two steps algorithm detailed in the paragraph 3.3. We perform a topological sorting of the clusters following the routine of the paragraph 4.1. Next, we draw the graph of each cluster as we extract its logic elements from the main graph `g`. Just like in `g`, we create one artificial node in the cluster's graph for each input signal of the cluster. The assessment of the cluster's logical function and the simulation of the simultaneous faults inside the cluster are described in the paragraph 4.2. Note that in the paragraph 4.2, the `bdd_ithvar` variables attached to each primary input of the clusters can be set thanks to the artificial nodes that we first drew in the cluster's graph. After that, we compute the SPR matrices chaining to obtain the SPR matrix of `PO` using the same process as the one we illustrated in the paragraph 2.2.2. All the primary input signals of `PO_subgraph` are considered to be equally distributed on the correct values $\{0_c, 1_c\}$, while their probability of being wrong is zero. So the SPR matrix of any input signal of `PO_subgraph` is:

$$\mathrm{SPR}_{\texttt{standard\_input}} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$$

5.2 CPBR Algorithm for ISCAS89 Benchmark

The protocol for processing the circuits from the ISCAS89 benchmark is a bit more complex, because these are sequential circuits. Nevertheless, if our tool is configured to handle such netlists, it will be much more convenient for us to adapt our tool for the analysis of processor netlists.

5.2.1 ISCAS89 Circuits Handling

We used the Verilog ISCAS89 gate netlists at the webpage [19]. The circuits of this benchmark are gate netlists, with a scope "DFF" called for each flip-flop instance. These sequential toggles make the combinatorial analysis more complex.

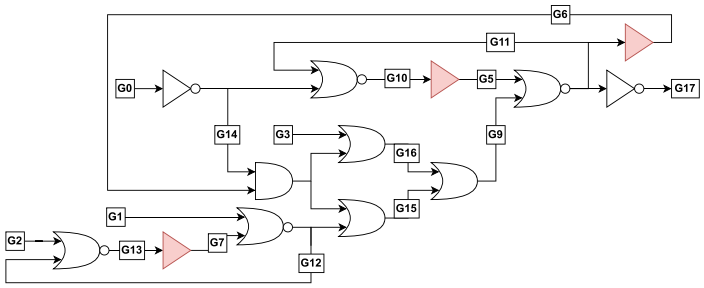
The first issue induced by the flip-flops is the sequential signals and nodes they imply, while we want to remain at a combinatorial level. To manage this we draw one coarsened logic node for each DFF scope. The attached BDD function is either a buffer or an inverter, depending on the output of the flip-flop used (Q or $\overline{Q}$). In the ISCAS89 benchmark, the flip-flops always stand for buffers. We represent the logic gates and the primary input signals of the Verilog netlist in the exact same way as we did for the ISCAS85 benchmark circuits. The resulting graph is called `g`.

The second issue induced by the toggles are the cycles that appear in the circuit graph `g`. If `g` is cyclic, there exist two logic nodes A and B in `g` such that the output signal of A is affected by the output signal of B and vice-versa. Thus, there is no way to obtain the SPR matrix of A's output without B's output, nor the contrary. To manage this problem, we make the circuit graph acyclic by cutting each output of each DFF in the graph. Thus, an output signal of a flip-flop becomes both a primary output and a primary input in the resulting graph. The latter is called `acyclic_g`. This is illustrated in Fig. 4a and b. Figure 4a depicts the graph `g` of the ISCAS89 benchmark's s27 circuit. All flip-flops are represented by a red buffer node in the circuit graph. Figure 4b depicts the graph `acyclic_g` of the s27 circuit. We broke every output edge from every flip-flop and changed the signals G5, G6 and G7 in both primary input and primary output signals.

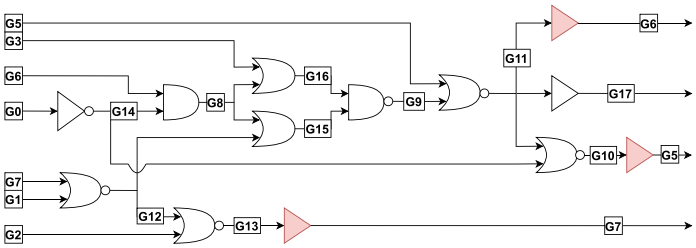
Next, we explain the entire CPBR process on ISCAS89 benhcmark circuits. We keep illustrating our statements with diagrams of the circuit s27, which is convenient since, thanks to its small size, there is no need to partition the netlist and we can explain more clearly the process applied to flip-flops.

In the presence of flip-flops in the circuit, the calculation of the correctness and error rates of the primary outputs of the circuit is done in two steps. First, we evaluate the error and the correctness rates of the flip-flops when only the flip-flops are subject to faults. Once we obtain the SPR matrices of the flip-flops, we assume that the success rates of the flip-flops against errors are fixed once and for all. The flip-flops are then considered as primary input signals of the circuit, with a non-zero probability of being incorrect, and we subsequently calculate the SPR matrices of the primary outputs of the circuit.

1st step We obtain the derivative circuit `acyclic_g` from `g` as we break the back-edges of the circuit as explained in the paragraph above. We loop over each flip-flop output signal which is a primary output of the graph `acyclic_g`. We call `FF_output` a node that generates a flip-flop output signal in `acyclic_g`. For any `FF_output`, we extract its cone of influence graph from `acyclic_g0` and we call it `FF_subgraph`. We partition `FF_subgraph` with the two steps algorithm detailed in the paragraph 3.3. We perform a



(a) ISCAS89/s27 circuit graph g, with coarsened buffer nodes instead of flip-flop instances



(b) ISCAS89/s27 `acyclic_g` graph, with coarsened buffer nodes instead of flip-flop instances

topological sorting of the clusters following the routine of the paragraph 4.1. The assessment of the clusters' logical function and the simulation of the simultaneous faults inside the cluster is roughly explained in the paragraph 4.2, but we add a subtlety there. In contrast to the paragraph 4.2, we only fault the logic nodes standing for flip-flops in the clusters, and not the ones standing for logic gates. Thus, the probability for a single output $\mathcal{S}$ of a cluster to be bit-flipped in this step is:

$$p_{\mathrm{err}_{\mathcal{S}}} = \sum_{k=1}^{k_{\max}} \left[\binom{N_F}{k} \sum_{j=1}^{2^N-1} \sum_{z=0} \mathcal{S}(z,0) \oplus \mathcal{S}(z, \mathcal{E}_k^j) \right] \times \frac{(1-q)^k q^{N_G+N_F-k}}{2^N \times \sum_{k=1}^{k_{\max}} \binom{N_F}{k}}$$

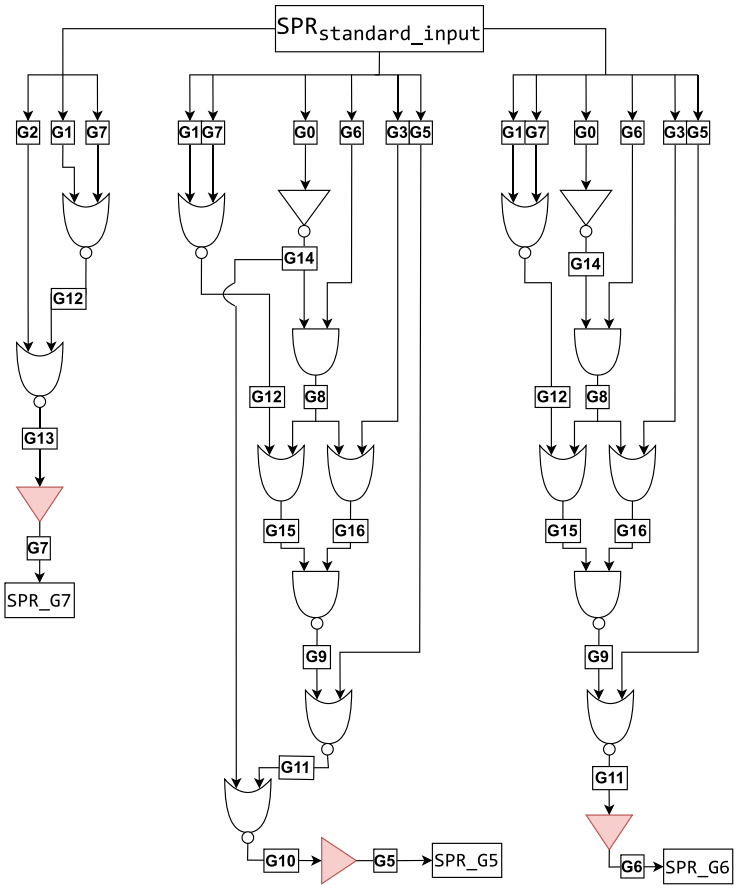
where N_F and N_G are respectively the number of flip-flop nodes and logic gate nodes in the cluster graph. All the other parameters of the equation are defined in the paragraph 2.1.1, assuming the parallel between the circuit and the cluster. Note that here, $k_{\max} \leq N_F$. Next, we compute the SPR matrices chaining to obtain the SPR matrix of `FF_output` with the same process as the one we illustrated in the paragraph 2.2.2. All the primary input signals of the graph `FF_subgraph` have a SPR matrix equal to $\mathrm{SPR}_{\texttt{standard\_input}}$.

This process is illustrated in Fig. 5a, where we selected the influence cones of the flip-flop nodes of the s27 circuit, and we computed their respective SPR matrices using the instructions from the current step.

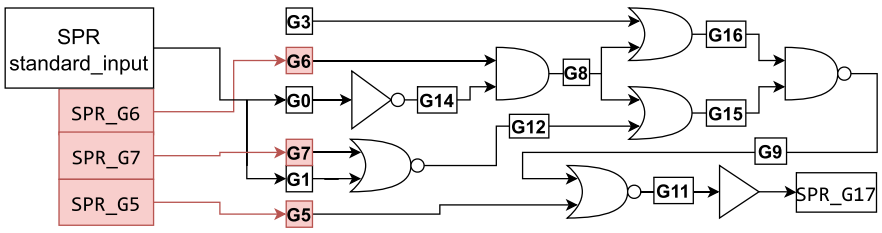
2nd Step Once we have the SPR matrices of all the flip-flop primary outputs of the graph `acyclic_g`, we consider the primary outputs of the circuit `g`. For each primary output `PO`, we extract the graph of the cone of influence of `PO` from `acyclic_g`. We call the extracted graph `PO_subgraph`. We refer again to the paragraphs 3.3 and 4.1 for its partitioning and for the topological sorting of the clusters. The simulation of the simultaneous faults inside the clusters follows the framework described in the paragraph 4.2, but here, in contrast to the previous step handling the flip-flops, only the nodes standing for logic gates can be faulted in the clusters. Thus, the probability for a single output $\mathcal{S}$ of a cluster to be bit-flipped in this step is:

$$p_{\mathrm{err}_{\mathcal{S}}} = \sum_{k=1}^{k_{\max}} \left[\binom{N_G}{k} \sum_{j=1}^{2^N-1} \sum_{z=0} \mathcal{S}(z,0) \oplus \mathcal{S}(z, \mathcal{E}_k^j) \right] \times \frac{(1-q)^k q^{N_G+N_F-k}}{2^N \times \sum_{k=1}^{k_{\max}} \binom{N_G}{k}}$$

And this time, $k_{\max} \leq N_G$. The nodes standing for the flip-flop instances cannot propagate erroneous output values. The SPR matrices chaining through the clusters remains the same. Finally, all the primary input signals that do not stand for a flip-flop output are represented by the SPR matrix $\mathrm{SPR}_{\texttt{standard\_input}}$, while all the primary input signals that stand for a flip-flop output are represented by the SPR



(a) The graphs of the cones of influence of the three flip-flops in the circuit s27



(b) The cone of influence graph of the primary output G17 of the circuit s27

Fig. 5 Influence cones of each flip-flop and of the output of circuit s27, with a SPR chaining illustration

matrix that was computed for their corresponding `FF_output` in the step 5.2.1.

This process is illustrated by Fig. 5b, where we select the influence cone of the primary output G17. We inject the SPR matrices of the output signals of the flip-flops to the corresponding primary inputs G5, G6 and G7. All other primary input signals are represented by $\mathrm{SPR}_{\texttt{standard\_input}}$. We obtain the SPR matrix of G17 from the CPBR algorithm by following the framework that we just described.

5.2.2 Note on the Computation of the Flip-Flops SPR Matrices

We use again the notations from the paragraph 5.2.1. For the sake of accuracy, we could have gone further in the step 5.2.1 by calculating a fixed point for the SPR matrix of `FF_output`. That would have consisted of the following recursive process. First we apply the algorithm described in the step 5.2.1. This gives a SPR matrix $\mathrm{SPR}^0_{\texttt{FF\_output}}$ for the output signal of the

flip-flop node `FF_output`. Next, instead of moving to the step 5.2.1, we repeat the process of the step 5.2.1 on the cone of influence of `FF_output`. However, this time, the primary input of the cone which corresponds to the output signal of `FF_output` is represented by $\mathrm{SPR}^0_{\texttt{FF\_output}}$, instead of $\mathrm{SPR}_{\texttt{standard\_input}}$. We repeat the recursion a large number of times, and at each iteration $n \in \mathbb{N}^*$, the primary input corresponding to the signal generated by `FF_output` is represented by $\mathrm{SPR}^{n-1}_{\texttt{FF\_output}}$, while the SPR matrix of that same signal computed during that step is called $\mathrm{SPR}^{n}_{\texttt{FF\_output}}$. Eventually, if the sequence $(\mathrm{SPR}^{n}_{\texttt{FF\_output}})_{n\in\mathbb{N}}$ converges as n tends to infinity, we can assert

$$\mathrm{SPR}_{\texttt{FF\_output}} = \lim_{n\to\infty} \mathrm{SPR}^{n}_{\texttt{FF\_output}}$$

We tried this method on the circuit s27, and only ten recursions were enough so that the SPR matrix of each flip-flop strives to a fixed point. We compared the correctness rates obtained on the flip-flops and the primary output G17, between the case where we use the method described in the paragraph 5.2.1, and the case where we add the fixed point calculation to the step 5.2.1. We obtained the following differences:

$$|\,\mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G7}}) - \mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G7},fp})| = 7.87e^{-4}$$
$$|\,\mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G6}}) - \mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G6},fp})| = 7.012387e^{-10}$$
$$|\,\mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G5}}) - \mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G5},fp})| = 7.545566e^{-10}$$
$$|\,\mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G17}}) - \mathrm{Tr}\,(\mathrm{SPR}_{\texttt{G17},fp})| = 1.64e^{-4}$$

It should be noted that the process of the paragraph 5.2.1 is equivalent to the case where we only do one iteration in the recursion of the fixed point process.

Although searching for the fixed point on flip-flop matrices is a more accurate and rigorous approach, we did not apply it in our CPBR C++ code, in order to avoid increasing the complexity of the algorithm. Nevertheless, it will be an interesting addition when the code is fully optimized.

5.3 Summary of the Full Process

To calculate a circuit's output signals correctness rates, we launch our CPBR C++ computer program once on the entire circuit. First, it constructs the graph of the netlist. We remind that for the circuits of the ISCAS85 benchmark, there is only one graph to build, but for the circuits of the

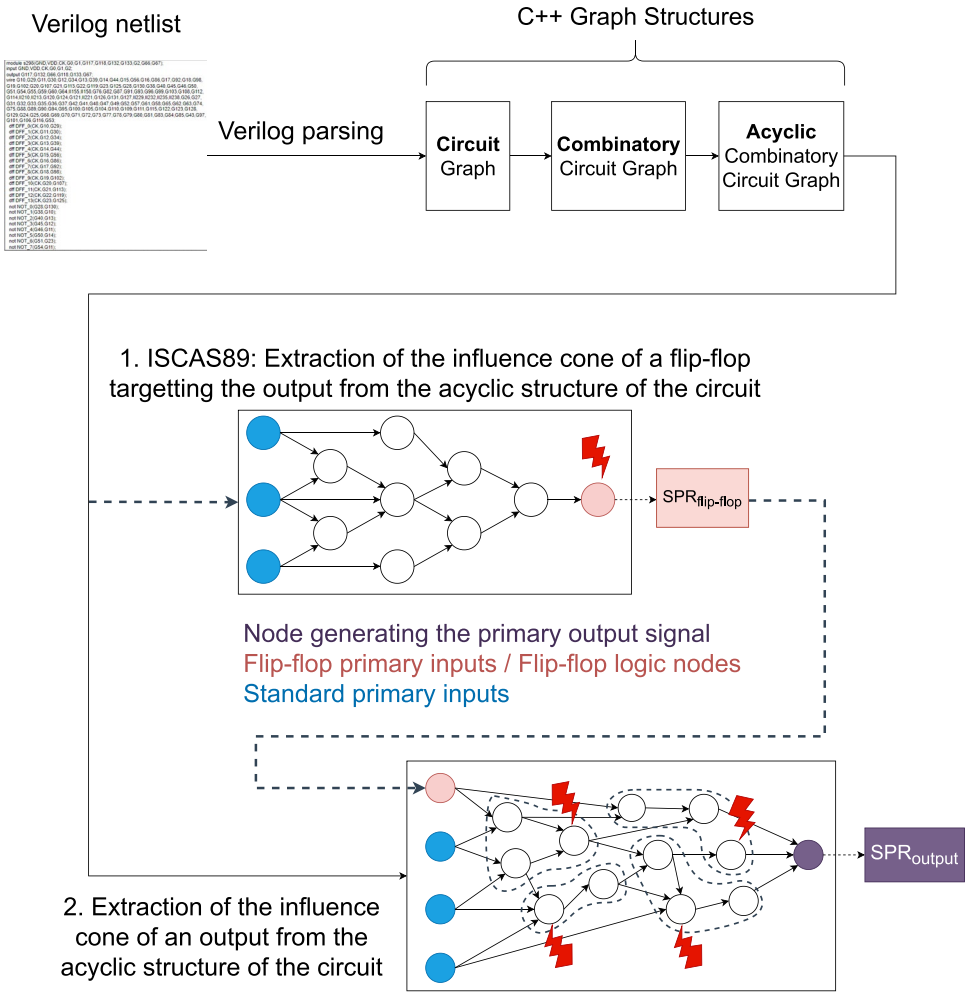


Fig. 6 Full process of CPBR on a circuit Verilog netlist

ISCAS89 benchmark, it is necessary to build both `g` and `acyclic_g`. In the case of the largest netlists, this step can take up to 5 min. Once the circuit graphs are built, the program reiterates the extraction of the cone of influence and the CPBR algorithm on this cone for each output, one by one. When processing circuits of the ISCAS89 benchmark, this is repeated first on each flip-flop output, and then secondly for each primary output. Thus, the step 5.2.1 is carried out correctly. This full process is summarized on Fig. 6.

6 Experimental Results on ISCAS85 and ISCAS89 Benchmarks

This section aims at conducting a quantitative study on the CPBR method. We analyze the computation times, memory consumption, and probabilistic rates produced by our tool on several circuits from the ISCAS85 and ISCAS89 benchmarks. In addition, we quantify the approximation made compared to the reference model by estimating the difference between the aforementioned values and the equivalent data for the PBR method.

6.1 Algorithm Parameters, Performances and Results

Algorithm Parameter We provide here the operating parameters of our computer program implementing the CPBR algorithm. All results of the CPBR method that follow in the document are generated using these parameters. Let C be a cluster in the partitioned circuit graph and let N_G and N_F be respectively the number of logic gates and flip-flops in C.

- The probability of malfunction of a logic node in the circuit is $(1 - q) = 0.05$.
- `NI = 10` is the maximum number of input signals in C.
- `NG = 35` is the maximum number of logic nodes in C.
- $k_{\max}$, the maximum number of simultaneous faults in C, depends on $N_{GF} \in \{N_G, N_F\}$ as written below:

$$k_{\max} = \begin{cases} 1 & \text{if } N_{GF} \in [\![0, 6]\!] \\ 2 & \text{if } N_{GF} \in [\![6, 9]\!] \\ 3 & \text{if } N_{GF} \in [\![10, 16]\!] \\ 4 & \text{if } N_{GF} \in [\![17, 25]\!] \\ 5 & \text{if } 26 \leq N_{GF} \end{cases} \tag{9}$$

We remind the reader that our work does not target the study of the most probable number of faults in a circuit or in a cluster. In this paper, we focus more on the propagation aspect, and as such, we set the parameter $k_{\max}$ accordingly to the time and memory complexity constraints of the CPBR method.

Performances Our CPBR C++ program is executed on computers of the Load Sharing Facility (LSF) cluster of STMicroelectronics at Crolles. The computers used are equipped with Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz processors. Tables 2 and 3 summarize the running times and the consumed memory of our CPBR C++ computer program on the circuits presented in Table 4, for the aforementioned parameters. For one designated circuit, the columns of these tables are organized as follows.

- The *Maximum Probabilistic Analyze Time Complexity (M.P.A.T.C.)* is an estimation of the CPBR method time complexity after the partitioning phase. It is assumed that we have sufficiently bounded the clusters in terms

Table 2 Summary of CPBR time complexity

Circuit	M.P.A.T.C.[a]	M.I.B.P.A.S.C.[b]	LSF resource usage summary		
			T.R.T.[c]	Maximum Memory	Average Memory
c2670	$\mathcal{O}(2^{29})$	$\mathcal{O}(2^{26})$	1579 s	78 MB	62.20 MB
c3540	$\mathcal{O}(2^{30})$	$\mathcal{O}(2^{23})$	13584 s	134 MB	89.12 MB
c5315	$\mathcal{O}(2^{29})$	$\mathcal{O}(2^{26})$	10518 s	147 MB	115.11 MB
c6288	$\mathcal{O}(2^{34})$	$\mathcal{O}(2^{23})$	87505 s	733 MB	436.32 MB
c7552	$\mathcal{O}(2^{30})$	$\mathcal{O}(2^{27})$	14615 s	166 MB	120.14 MB
s5378	$\mathcal{O}(2^{30})$	$\mathcal{O}(2^{26})$	7223 s	128 MB	113.84 MB
s9234	$\mathcal{O}(2^{14})$	$\mathcal{O}(2^{24})$	8792 s	212 MB	143.71 MB
s13207	$\mathcal{O}(2^{29})$	$\mathcal{O}(2^{26})$	29292 s	261 MB	234.49 MB
s15850	$\mathcal{O}(2^{26})$	$\mathcal{O}(2^{26})$	75279 s	718 MB	586.77 MB
s35932	$\mathcal{O}(2^{8})$	$\mathcal{O}(2^{17})$	70580 s	372 MB	193.88 MB
s38417	$\mathcal{O}(2^{14})$	$\mathcal{O}(2^{24})$	169804 s	997 MB	734.47 MB
s38584	$\mathcal{O}(2^{14})$	$\mathcal{O}(2^{26})$	139522 s	668 MB	639.59 MB

[a]Maximum Probabilistic Analyze Time Complexity

[b]Maximal Inferior Bound on the Probabilistic Analyze Size Complexity

[c]Total Running Time

Table 3 Longest run of CPBR on an output for the processed circuits

Circuit	Longest Run on 1 Output	Longest Output to be Processed	N[a]	N_F[b]	N_G[c]
c2670	463 s	G2594	122	0	828
c3540	2588 s	N5360	50	0	1458
c5315	772 s	N8127	67	0	937
c6288	5364 s	N6287	32	0	2325
c7552	1349s	N10729	194	0	1096
s5378	626 s	n3144gat	56	0	455
s9234	511 s	g127	82	1	1005
s13207	2693 s	g9378	212	0	1191
s15850	345 s	g10455	31	0	335
s35932	78 s	WX7126	14	1	87
s38417	559 s	g2300	99	1	471
s38584	463 s	g2955	84	1	450

[a]Number of input signals in the cone of influence of the output

[b]Number of flip-flop logic nodes in the cone of influence of the output

[c]Number of logic gates in the cone of influence of the output

of the number of logical nodes and the number of input signals so that matrix calculations (standard products and Kronecker products) can be considered constant in time, that is, with a complexity of $\mathcal{O}(1)$. In this configuration, the temporal complexity of the CPBR algorithm after partitioning is approximately equal to the sum of the temporal complexities of the PBR analysis in each cluster. From this axiom, we formalize the temporal complexity of the CPBR algorithm post-partitioning. Let $\mathcal{C}$irc be a circuit, let $\mathcal{Q}$ be an output of $\mathcal{C}$irc, and let $\{C_1^{\mathcal{Q}}, \ldots, C_n^{\mathcal{Q}}\}$, with $n \in \mathbb{N}^*$, be the clusters list resulting from the partitioning algorithm applied to $\mathcal{Q}$'s influence cone. The time complexity of PBR analysis in a cluster $C_k^{\mathcal{Q}}$, with $k \in [\![1, n]\!]$, is written $\mathcal{T}_{\text{PBR}}\left(\mathcal{C}\text{irc}, C_k^{\mathcal{Q}}\right)$, and is defined in (3). The time complexity of the CPBR algorithm after partitioning applied to the influence cone of $\mathcal{Q}$ is then:

$$\mathcal{T}_{\text{CPBR}}(\mathcal{C}\text{irc}, \mathcal{Q}) = \sum_{k=1}^{n} \mathcal{T}_{\text{PBR}}\left(\mathcal{C}\text{irc}, C_k^{\mathcal{Q}}\right) \tag{10}$$

and we set:

$$\text{M.P.A.T.C.} := max\ \{\mathcal{T}_{\text{CPBR}}(\mathcal{C}\text{irc}, \mathcal{Q}) \mid \mathcal{Q} \text{ a primary output of } \mathcal{C}\text{irc}\}$$

- The *Maximal Inferior Bound on the Probabilistic Analyze Size Complexity (M.I.B.P.A.S.C.)* is an estimation of the inferior bound of the CPBR method size complexity after the partitioning phase. We put this concept in parallel with the lower bound of the memory complexity of the PBR method, given in (4). We formalize the definition of this concept in mathematical terms by using the notations from the previous item. An inferior bound on the memory complexity of the CPBR algorithm applied to the output $\mathcal{Q}$ is:

$$\mathcal{I}nf\ \mathcal{S}_{\text{CPBR}}(\mathcal{C}\text{irc}, \mathcal{Q}) = \sum_{k=1}^{n} \mathcal{I}nf\ \mathcal{S}_{\text{PBR}}(\mathcal{C}\text{irc}, C_k^{\mathcal{Q}}) \tag{11}$$

and we set:

$$\text{M.I.B.P.A.S.C.} := max\ \{\mathcal{I}nf\ \mathcal{S}_{\text{CPBR}}(\mathcal{C}\text{irc}, \mathcal{Q}) \mid \mathcal{Q} \text{ a primary output of } \mathcal{C}\text{irc}\}$$

- The *Total Running Time (T.R.T.)* is the running time of our CPBR program implementing the full process described in paragraph 5.3, i.e. the iteration of the CPBR algorithm on the influence cone of each primary output of the circuit, one by one.
- The *Longest Run on one Output* is the maximum calculation time for one output of the circuit.

Table 4 Parameters of the circuits studied in Tables 3, 4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, and 19

Benchmark	Circuit	N[a]	N_F[b]	N_G[c]	N_O[d]
ISCAS85	c2670	157	0	1181	64
	c3540	50	0	1669	22
	c5315	178	0	2307	123
	c6288	32	0	2416	32
	c7552	207	0	3513	108
ISCAS89	s5378	35	179	2779	49
	s9234	36	211	5597	39
	s13207	62	638	7951	152
	s15850	77	534	9772	150
	s35932	35	1728	16065	320
	s38417	28	1636	22179	106
	s38584	38	1426	19253	304

[a]Number of input signals in the designated circuit

[b]Number of flip-flops in the designated circuit

[c]Number of logic gates in the designated circuit

[d]Number of output signals in the designated circuit

Correctness Rates Results Tables 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, and 19 of Annex give the correctness rates that we obtained on some outputs of the circuits presented in Table 2. There is no point in displaying hundreds of probability values here, so we only give the results of a few output signals of the circuits that we have calculated.

Table 5 Parameters of the three smallest influence cones of the circuit ISCAS85/c432

Output	N[a]	N_G[b]
N223	18	20
N329	27	58
N370	36	105

[a]Number of input signals in the cone of influence of the output

[b]Number of logic gates in the cone of influence of the output

Table 6 Evolution of memory and time consumption of the PBR algorithm on the influence cones of the outputs N223, N329 and N370 of the circuit ISCAS85/c432

Output	k_{max}[a]	$\mathcal{T}_{PBR}$[b]	$\mathcal{I}nf\ \mathcal{S}_{PBR}$[c]	Completion Status	LSF resource usage summary		
					Running Time	Maximum Memory	Average Memory
N223	1	$\mathcal{O}(2^{22})$	$\mathcal{O}(2^{9})$	Success	36 s	11 MB	8.5 MB
	2	$\mathcal{O}(2^{26})$	$\mathcal{O}(2^{12})$	Success	39 s	12 MB	9.0 MB
	3	$\mathcal{O}(2^{28})$	$\mathcal{O}(2^{15})$	Success	37 s	33 MB	9.8 MB
	4	$\mathcal{O}(2^{31})$	$\mathcal{O}(2^{17})$	Success	74 s	33 MB	19.8 MB
N329	1	$\mathcal{O}(2^{33})$	$\mathcal{O}(2^{12})$	Success	9357 s	5393 MB	2995.15 MB
	2	$\mathcal{O}(2^{38})$	$\mathcal{O}(2^{17})$	Success	22286 s	5403 MB	4448.47 MB
	3	$\mathcal{O}(2^{42})$	$\mathcal{O}(2^{21})$	Failure	300756 s	5403 MB	5223.10 MB
	4	$\mathcal{O}(2^{46})$	$\mathcal{O}(2^{25})$	Failure	92082 s	5403 MB	5075.97 MB
N370	1	$\mathcal{O}(2^{43})$	$\mathcal{O}(2^{13})$	Failure	154036 s	22712 MB	15549.89 MB
	2	$\mathcal{O}(2^{48})$	$\mathcal{O}(2^{19})$	Failure	77190 s	15676 MB	10621.30 MB
	3	$\mathcal{O}(2^{54})$	$\mathcal{O}(2^{24})$	Failure	42899 s	11972 MB	8311.67 MB
	4	$\mathcal{O}(2^{58})$	$\mathcal{O}(2^{29})$	Failure	19310 s	8311 MB	5780.62 MB

[a]Number of simultaneous faults in the cone of influence of the output for the PBR algorithm

[b]Time complexity of the PBR approach given in (3)

[c]Inferior bound on the size complexity of the PBR approach given in (4)

Those correctness rates correspond to the aforementioned parameters for the CPBR algorithm. Those tables also provide more precise information on the running time of our program when it processes one output at a time. To explain these information, suppose $\mathcal{S}$ is a circuit output.

- The *Cone of Influence Extraction Time (CoI E.T.)* is the time required to extract the graph of the cone of influence of $\mathcal{S}$, from the graph of the entire circuit.
- The *Partitioning Time* is the time required to partition the cone of influence of $\mathcal{S}$, with the algorithm of the paragraph 3.3.
- The *Propagation Time* is the time required to build the topological order of the clusters, and to calculate all the ITM, PTM and SPR matrices of the clusters of the cone of influence of $\mathcal{S}$.

6.2 Complexity Comparison with PBR

In [14], we provided some examples of rates comparisons between the PBR and the CPBR models, and between several partitions when applying CPBR, on several multiplier designs.

With Tables 6 and 7, we provide new comparisons on memory and time performances, when applying either PBR or CPBR on the three smallest influence cones of the circuit c432, from the ISCAS85 benchmark. The parameters of the later cones are given in Table 5. We produced a generic implementation of the PBR algorithm, which is based, like CPBR, on the use of BDDs, and that can be parameterized with arbitrary values of k_{max}. Table 6 shows the evolution of memory consumption and execution time of our PBR computer program, according to the number of inputs, the number of logical gates and the number of simultaneous faults inside the targeted influence cone. When the completion failed, it is written "Failure" in the Completion Status column. In this case, the data in the Running Time column corresponds to the duration of the execution until the program crashed, and the Maximum Memory and Average Memory data are sent by LSF at the moment when the program stopped. Most of the time, the program's failure is due to a memory issue, and it ends with the error code "Bus error". Table 7 shows the performances of our CPBR program on the same influence cones, with k_{max} set in the clusters according to the parameterization in (9).

Table 7 CPBR performances on the influence cones of the outputs N223, N329 and N370 of the circuit ISCAS85/c432, for the parametrization of k_{max} provided in (9)

Output	$\mathcal{T}_{CPBR}$[a]	$\mathcal{I}nf\ \mathcal{S}_{CPBR}$[b]	LSF resource usage summary		
			Running Time	Maximum Memory	Average Memory
N223	$\mathcal{O}(2^{12})$	$\mathcal{O}(2^{6})$	47 s	11 MB	8.8 MB
N329	$\mathcal{O}(2^{15})$	$\mathcal{O}(2^{9})$	32 s	11 MB	8.25 MB
N370	$\mathcal{O}(2^{18})$	$\mathcal{O}(2^{12})$	33 s	12 MB	9.25 MB

[a]Probabilistic analyze time complexity of the CPBR algorithm after partitioning, given in (10)

[b]Inferior bound on the probabilistic analyze size complexity of the CPBR algorithm after partitioning given in (11)

We observe that before a certain threshold value for the (N, N_G) pairs in Table 6, PBR is more effective than CPBR. More precisely, PBR is faster and consumes less memory than CPBR until we set $k_{\max}$ to 4 when processing N223. This is because CPBR has more calculations to perform, for the search of an adequate partition, for the matrix multiplications between clusters, etc. while PBR only performs one analytical calculation of a largely reasonable complexity. Nevertheless, the PBR program already struggles to handle the output N329 that involves 27 input signals and 58 logic gates, while CPBR completes the calculation in only 32 s. The average memory consumed by PBR exceeds the one consumed by CPBR starting from $k_{\max} = 3$ in N223's influence cone. At the beginning, the gap is not that significant, since the largest delta in the average memory consumed by the to programs in N223's influence cone is 11 MB and occurs when $k_{\max} = 4$. However, the deltas in the average memory and in the execution time skyrocket from the first PBR test on N329's influence cone. Indeed, when $k_{\max} = 1$, the PBR program execution consumed about 2987 MB and lasted about two hours and a half more than the CPBR program. These pieces of information are displayed in a more impactful visual way in Fig. 7. This figure shows the temporal (respectively, memory) consumption evolution of the PBR method as a function of $k_{\max}$ with, in the background, the temporal (respectively, memory) consumption of CPBR depending on the parameterization of $k_{\max}$ in (9). We can see that the two curves related to N329's influence cone show a decrease for $k_{\max} = 4$. This is due to the fact that, starting from 3 simultaneous errors in N329's cone, our PBR program failed to execute the algorithm in its entirety. The program is ending more and more prematurely relative to the amount of operations it has to perform. As a consequence, the memory consumption indicated in Table 6 decreases from $k_{\max} = 3$ for N329's cone, and decreases significantly with respect to $k_{\max}$ when processing N370's cone. In addition to these comparisons, a striking perk of CPBR is not mentioned in the tables: the T.R.T. of our CPBR program on the c432 circuit is 1 min, which is only 23 s more than the time taken by our PBR program to process the sole output N223 when $k_{\max} = 3$.

Since we target the failure rate analysis of much larger circuits than the c432, which is only the second smallest circuit of the ISCAS85 benchmark, it is straightforward that the use of CPBR is much more relevant and adapted.

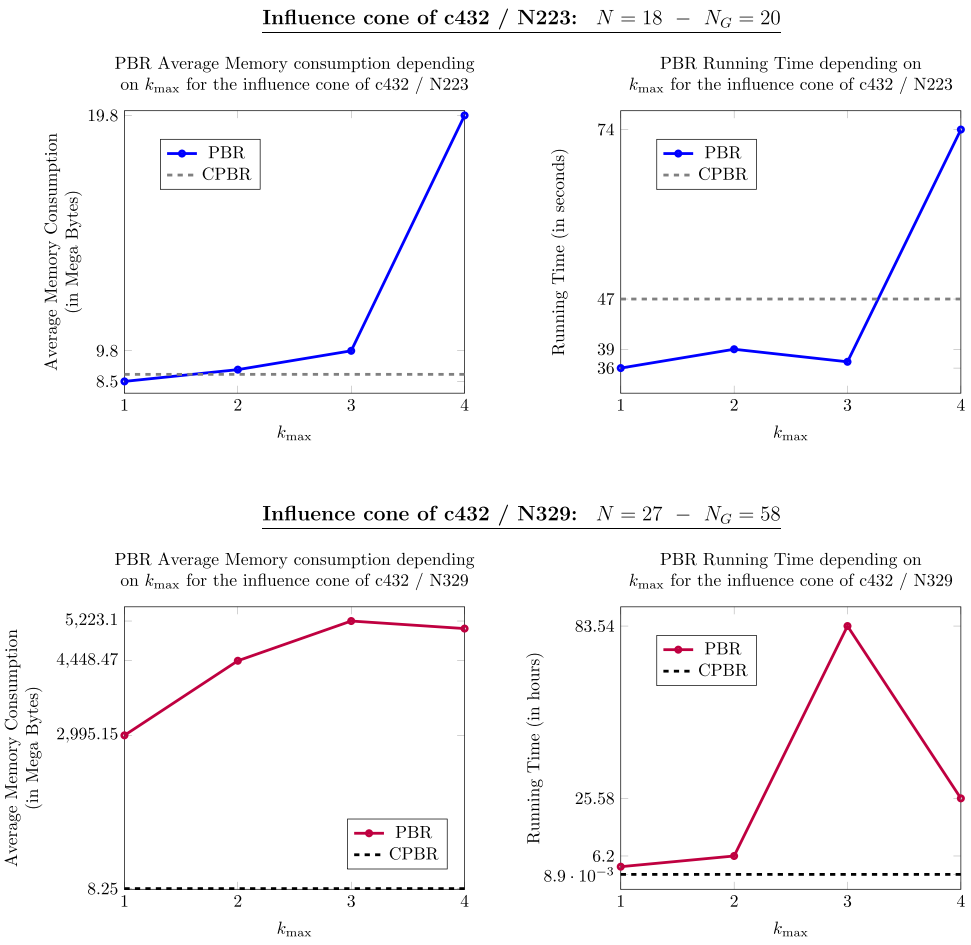


Fig. 7 Comparison between CPBR and PBR on time and memory consumption through curve plots

6.3 Discussion of Results

Our results show that even on non-arithmetic circuits, our method is scalable and our fault model produces remarkably high correctness rates, that ensure good circuit categorization according to *Automotive Safety Integrity Level (ASIL)* [20] standards. Thus, the CPBR method is very effective compared to the simple PBR model.

In some netlists, the complexity issues of our CPBR C++ computer program is exacerbated by the number of output signals to be processed. So we are working on parallelizing our C++ code. In ideal circumstances, we would calculate in parallel (1) the SPR matrices of all the flip-flops, (2) then the SPR matrices of all the primary outputs. In that case, the complexity of the CPBR algorithm will not be the sum of the complexities of all the clusters in all the cones of influence, but rather the addition of the complexities of the two heaviest clusters, in the first step and in the second step.

Unfortunately, our CPBR C++ computer program is currently compiled inside the Icarus environment, which does not support multithreading. So we aim at dividing the CPBR algorithm in two programs. The first one would remain inside the Icarus environment to extract the circuits graph structures, while the second one would implement the CPBR algorithm with parallelization in a usual `gcc` environment.

7 Conclusion and Future Work

We extended the work of the CPBR model as we implemented a three-steps generic algorithm. The first step is the conversion of a Verilog netlist to a graph. The second step is the implementation of the generic partitioning algorithm, for the circuits of both the ISCAS85 and the ISCAS89 benchmarks. In the third step, we reproduce the clusters logic functions in a very generic way, with and without the simulation of the errors, and we propagate the error rates from clusters to clusters with SPR matrices. This was allowed by the BDDs that are quite easy to handle thanks to the BuDDy library, which simulates the logic nodes well enough.

The next step in our research is the further optimization of our algorithm to make it faster on significant netlists. We are working both on optimizing the code's own functions and on parallelization to calculate multiple outputs at once. Eventually, we aim at processing combinatorial netlists composed of gate libraries elements such as Synopsys *GTECH* or STMicroelectronics *FDSOI 28nm*. A library element, such as a *nand* gate, appears as a scope in Icarus internal representation, which requires further support in our tool to extract its PTM and ITM in a generic and portable way. Once such libraries are supported, our tools will be applicable to a very wide range of netlists for safety analysis.

Appendix

Some Results on Benchmarks Circuits ISCAS85 and ISCAS89

Table 8 c2670

Output	N	N_G	Correctness Rate	CoI E.T.[a]	Partitioning Time	Propagation Time
G2588	35	122	0.893579	5 s	2 s	1 s
G2590	42	137	0.94082	5 s	4 s	1 s
G2594	122	828	0.999984	38 s	217 s	207 s

[a]Cone of Influence Extraction Time

Table 9 c3540

Output	N	N_G	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
N5192	49	1380	0.999997	69 s	1046 s	1434 s
N5231	49	1386	0.980463	68 s	1021 s	1423 s
N5361	49	1433	1	66 s	1041 s	1426 s

Table 10 c5315

Output	N	N_G	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
N8075	32	436	0.806326	28 s	77 s	92 s
N8123	66	933	0.894172	66 s	435 s	230 s
N8128	67	937	0.927528	77 s	442 s	237 s

Table 11 c6288

Output	N	N_G	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
N6170	32	1571	0.999995	114 s	605 s	2523 s
N6190	32	1797	0.998613	135 s	807 s	2829 s
N6288	32	2327	0.999947	188 s	1732 s	3419 s

Table 12 c7552labeltab:c7552

Output	N	N_G	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
N10706	94	499	0.975437	44 s	75 s	323 s
N10729	194	1096	0.999997	104 s	472 s	773 s
N11334	94	676	0.853495	66 s	184 s	509 s

Table 13 s5378

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
n3109gat	5	26	0	0.851027	1 s	0 s	3 s
n3143gat	61	476	0	0.99852	34 s	31 s	510 s
n3144gat	56	455	0	0.977675	31 s	29 s	564 s

Table 14 s9234

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
g6284	21	169	0	0.994766	30 s	3 s	80 s
g6372	14	140	0	0.949258	23 s	2 s	36 s
g4121	2	15	0	0.999997	2 s	0 s	0 s

Table 15 s13207

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
g9310	32	404	0	0.999998	178 s	82 s	564 s
g6236	2	11	0	0.999536	4 s	0 s	0 s
g1724	1	2	0	0.9525	0 s	0 s	0 s

Table 16 s15850

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
g10379	19	294	0	0.987239	91 s	20 s	88 s
g2986	1	2	0	0.9525	0 s	0 s	0 s
g8986	2	35	0	1	9 s	0 s	21 s

Table 17 s35932

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
CRC_OUT_4_13	3	8	1	0.988125	9 s	0 s	0 s
DATA_9_6	6	34	0	0.871764	29 s	0 s	23 s
DATA_9_0	6	34	0	0.908837	29 s	0 s	23 s

Table 18 s38417

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
g5629	1	2	0	0.9525	3 s	0 s	0 s
g6750	1	2	0	0.9525	3 s	0 s	0 s
g16496	3	9	0	0.703125	9 s	0 s	0 s

Table 19 s38584

Output	N	N_G	N_F	Correctness Rate	CoI E.T.	Partitioning Time	Propagation Time
g34235	13	288	0	0.998417	261 s	27 s	1063 s
g34240	13	286	0	0.999935	274 s	29 s	1002 s
g34236	13	288	0	0.964334	284 s	33 s	1105 s

Acknowledgements The authors whish to thank Thomas Fourier for his careful proofreading and technical advices. He was an intern student at STMicroelectronics from June to August 2023, during his second year at École Polytechnique, Palaiseau, France.

Funding This work was supported by Télécom Paris (Palaiseau, France), and STMicroelectronics (Crolles, France). The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data Availability The Verilog benchmarks ISCAS85 and ISCAS89 are available at [19] online.

Declarations

Conflicts of Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Auth C, Aliyarukunju A, Asoro M, Bergstrom D, Bhagwat V, Birdsall J, Bisnik N, Buehler M, Chikarmane V, Ding G, Fu Q, Gomez H, Han W, Hanken D, Haran M, Hattendorf M, Heussner R, Hiramatsu H, Ho B, Jaloviar S, Jin I, Joshi S, Kirby S, Kosaraju S, Kothari H, Leatherman G, Lee K, Leib J, Madhavan A, Marla K, Meyer H, Mule T, Parker C, Parthasarathy S, Pelto C, Pipes L, Post I, Prince M, Rahman A, Rajamani S, Saha A, Santos JD, Sharma M, Sharma V, Shin J, Sinha P, Smith P, Sprinkle M, Amour AS, Staus C, Suri R, Towner D, Tripathi A, Tura A, Ward C, Yeoh A (2017) A 10nm high performance and low-power CMOS technology featuring 3rd generation FINFET transistors, self-aligned quad patterning, contact over active gate and cobalt local interconnects. Proceedings of 2017 IEEE International Electron Devices Meeting (IEDM). pp 29–112914. https://doi.org/10.1109/IEDM.2017.8268472
2. Bottoni C, Coeffic B, Daveau J-M, Gasiot G, Naviner LADB, Roche P (2015) A layout-aware approach to fault injection for improving failure mode prediction. Proceedings of Workshop on Silicon Errors in Logic - System Effects (SELSE), Austin, United States. https://www.researchgate.net/publication/277140022_A_Layout-Aware_Approach_to_Fault_Injection_for_Improving_Failure_Mode_Prediction
3. Cai J, Chen C (2017) Circuit reliability analysis using signal reliability correlations. Proceedings of 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). pp 171–176. https://doi.org/10.1109/QRS-C.2017.34
4. Cai S, He B, Wang W (2020) Soft error reliability evaluation of nanoscale logic circuits in the presence of multiple transient faults. J Electron Test 36:469–483. https://doi.org/10.1007/s10836-020-05898-x
5. Cai S, He B, Wu S (2022) An accurate estimation algorithm for failure probability of logic circuits using correlation separation. J Electron Test 38:165–180. https://doi.org/10.1007/s10836-022-05996-y
6. Chen C, Cai J, Zhan S (2018) A triple-point model for circuit-level reliability analysis. Proceedings of 2018 IEEE International Symposium on Circuits and Systems (ISCAS). pp 1–4. https://doi.org/10.1109/ISCAS.2018.8350987
7. Coeffic B, Daveau J-M, Gasiot G, Pricco AE, Parini S, Scholastique T, Naviner LAB, Roche P (2016) Radiation hardening improvement of a SerDes under heavy ions up to 60 MeV.cm2/mg by layout-aware fault injection. Proceedings of IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE), Austin, Texas, United States. https://www.researchgate.net/publication/299603733_Radiation_Hardening_Improvement_

of_a_SerDes_under_Heavy_Ions_up_to_60_MeVcm2mg_by_Layout-Aware_Fault_Injection
8. de Vasconcelos MCR, Franco DT, Naviner DB, Lirida A, Naviner J-F (2008) Reliability analysis of combinational circuits based on a probabilistic binomial model. Proceedings of 2008 Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference. pp 310–313. https://doi.org/10.1109/NEWCAS.2008.4606383
9. Farias CR, Schvittz RB, Balen TR, Butzen PF (2022) Evaluating soft error reliability of combinational circuits using a Monte Carlo based method. Proceedings of 2022 IEEE 23rd Latin American Test Symposium (LATS). pp 1–6. https://doi.org/10.1109/LATS57337.2022.9936911
10. Flaquer JT, Daveau JM, Naviner L, Roche P (2010) Fast reliability analysis of combinatorial logic circuits using conditional probabilities. Microelectron Reliab 50(9):1215–1218. https://doi.org/10.1016/j.microrel.2010.07.058. 21st European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis
11. Flaquer JT, Daveau JM, Naviner L, Roche P (2011) An approach to reduce computational cost in combinatorial logic netlist reliability analysis using circuit clustering and conditional probabilities. Proceedings of 2011 IEEE 17th International On-Line Testing Symposium. pp 98–103. https://doi.org/10.1109/IOLTS.2011.5993818
12. Franco DT, Vasconcelos MC, Naviner L, Naviner J-F (2008) Reliability of logic circuits under multiple simultaneous faults. Proceedings of 2008 51st Midwest Symposium on Circuits and Systems. pp 265–268. https://doi.org/10.1109/MWSCAS.2008.4616787
13. Franco DT, Correia Vasconcelos M, Naviner L, Naviner J-F (2008) Reliability analysis of logic circuits based on signal probability. Proceedings of 2008 15th IEEE International Conference on Electronics, Circuits and Systems. pp 670–673. https://doi.org/10.1109/ICECS.2008.4674942
14. Goudet E, Treviño LP, Naviner L, Daveau J-M, Roche P (2023) Fast analysis of combinatorial netlists correctness rate based on binomial law and partitioning. Proceedings of 2023 IEEE 24th Latin American Test Symposium (LATS). pp 1–6. https://doi.org/10.1109/LATS58125.2023.10154491
15. Goudet E, Naviner L, Daveau J-M, Roche P (2023) Fast and accurate estimation of correctness rate in combinatorial circuits based on clustering. Proceedings of the 19th IEEE Workshop on Silicon Errors in Logic–System Effects (SELSE). pp 1–6
16. Herrmann J, Özkaya YM, Uçar B, Kaya K, Ümit V (2019) Çatalyürek: Multilevel algorithms for acyclic partitioning of directed acyclic graphs. SIAM J Sci Comput 41(4):2117–2145. https://doi.org/10.1137/18M1176865
17. BuDDy C++ library. https://buddy.sourceforge.net/manual/main.html. Online; Accessed 5 May 2024
18. https://github.com/steveicarus/iverilog. Accessed 5 May 2024
19. https://pld.ttu.ee/~maksim/benchmarks/. Accessed 5 May 2024
20. https://www.arteris.com/blog/top-35-iso-26262-acronyms-andabbreviations#:~:text=The%20single%20point%20fault%20metric,the%20other%20hardware%20architectural%20metric. Accessed 5 May 2024
21. https://www.boost.org/doc/libs/1_80_0/libs/graph/doc/index.html. Accessed 5 May 2024
22. ISO (2018) Road vehicles - functional safety. https://www.iso.org. Accessed 5 May 2024
23. Jahanirad H (2019) CC-SPRA: correlation coefficients approach for signal probability-based reliability analysis. IEEE Trans Very Large Scale Integr VLSI Syst 27(4):927–939. https://doi.org/10.1109/TVLSI.2018.2886027
24. Jahanirad H, Hosseini M (2020) Reliability estimation of CNT-FET-based combinational logic circuits. Proceedings of 2020 28th Iranian Conference on Electrical Engineering (ICEE). pp 1–5. https://doi.org/10.1109/ICEE50131.2020.9260712
25. Jamil M, Mukhopadhay S, Ghoneim M, Shailos A, Prasad C, Meric I, Ramey S (2023) Reliability studies on advanced FINFET transistors of the intel 4 CMOS technology. Proceedings of 2023 IEEE International Reliability Physics Symposium (IRPS). pp 1–5. https://doi.org/10.1109/IRPS48203.2023.10117992
26. Krishnaswamy S, Viamontes GF, Markov IL, Hayes JP (2005) Accurate reliability evaluation and enhancement via probabilistic transfer matrices. Proceedings of Design, Automation and Test in Europe (DATE). pp 282–2871. https://doi.org/10.1109/DATE.2005.47
27. Liu S-E, Li J, Nayak D, Marathe A, Balamukundhan K, Gosavi V, Prajapati A, Kilic B, Pang M, Mittal A (2022) Reliability qualification challenges of SOCS in advanced CMOS process nodes (invited). Proceedings of 2022 IEEE International Reliability Physics Symposium (IRPS). pp 8–11816. https://doi.org/10.1109/IRPS48227.2022.9764426
28. Marques EC, Paiva NM, Naviner LAB, Naviner J-F (2010) A new fault generator suitable for reliability analysis of digital circuits. Proceedings of 2010 Argentine School of Micro-Nanoelectronics, Technology and Applications (EAMTA). pp 41–45
29. Mohammadi K, Jahanirad H, Attarsharghi P (2011) Fast reliability analysis method for sequential logic circuits. Proceedings of 2011 21st International Conference on Systems Engineering. pp 352–356. https://doi.org/10.1109/ICSEng.2011.70
30. Normand E, Wert JL, Quinn H, Fairbanks TD, Michalak S, Grider G, Iwanchuk P, Morrison J, Wender S, Johnson S (2010) First record of single-event upset on ground, cray-1 computer at Los Alamos in 1976. IEEE Trans Nucl Sci 57(6):3114–3120. https://doi.org/10.1109/TNS.2010.2083687
31. O'Gorman TJ, Ross JM, Taber AH, Ziegler JF, Muhlfeld HP, Montrose CJ, Curtis HW, Walsh JL (1996) Field testing for cosmic ray soft errors in semiconductor memories. IBM J Res Dev 40(1):41–50. https://doi.org/10.1147/rd.401.0041
32. Papa D, Markov I (2007) Handbook of approximation algorithms and metaheuristics. https://doi.org/10.1201/9781420010749.ch61
33. Patel KN, Markov IL, Hayes JP (2003) Evaluating circuit reliability under probabilistic gate-level fault models. Proceedings of the International Workshop on Logic and Synthesis (IWLS). pp 59–64. https://api.semanticscholar.org/CorpusID:6600768
34. Pontes MF, Butzen PF, Schvittz RB, Rosa SL, Franco DT (2018) The suitability of the SPR-MP method to evaluate the reliability of logic circuits. Proceedings of 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS). pp 433–436. https://doi.org/10.1109/ICECS.2018.8617852
35. Pontes M, Farias CR, Schvittz R, Butzen P, Rosa LS (2021) Survey on reliability estimation in digital circuits. J Integr Circuits Syst. https://doi.org/10.29292/jics.v16i3.568
36. Pontes MF, Oliveira IFV, Schvittz RB, Rosa LS, Butzen PF (2022) The impact of logic gates susceptibility in overall circuit reliability analysis. Proceedings of 2022 IEEE International Symposium on Circuits and Systems (ISCAS). pp 1610–1614. https://doi.org/10.1109/ISCAS48785.2022.9937573
37. Popp M, Schlag S, Schulz C, Seemaier D (2021) Multilevel acyclic hypergraph partitioning. 2021 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX). pp 1–15. https://doi.org/10.1137/1.9781611976472.1
38. Roy TR, Sadi MS (2021) An efficient approach to tolerate soft errors in combinational circuits. Proceedings of 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)(ICICCS). pp 648–655. https://doi.org/10.1109/ICICCS51141.2021.9432122
39. Schivittz RB, Franco DT, Meinhardt C, Butzen PF (2016) A probabilistic model for stuck-on faults in combinational logic gates. Proceedings of 2016 17th Latin-American Test Symposium (LATS). https://doi.org/10.1109/LATW.2016.7483337
40. Schlag S, Henne V, Heuer T, Meyerhenke H, Sanders P, Schulz C (2016) k-way hypergraph partitioning via n-level recursive

bisection. 2016 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX). pp 53–67. https://doi.org/10.1137/1.9781611974317.5
41. Srinivasu B, Sridharan K (2017) A transistor-level probabilistic approach for reliability analysis of arithmetic circuits with applications to emerging technologies. IEEE Trans Reliab 66(2):440–457. https://doi.org/10.1109/TR.2016.2642168
42. Stempkovskiy A, Telpukhov D, Nadolenko V (2019) Accurate soft error rate reduction using modified resolution method. Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS). pp 1–6. https://doi.org/10.1109/EWDTS.2019.8884417
43. Teixeira Franco D (2008) Fiabilité du signal des circuits logiques combinatoires sous fautes simultanées multiples. Theses, Télécom ParisTech. https://pastel.archives-ouvertes.fr/pastel-00005125. Accessed 5 May 2024
44. Torras Flaquer J, Daveau J-M, Naviner L, Roche P (2010) Handling reconvergent paths using conditional probabilities in combinatorial logic netlist reliability estimation. Proceedings of 2010 17th IEEE International Conference on Electronics, Circuits and Systems. pp 263–267. https://doi.org/10.1109/ICECS.2010.5724504
45. Torras Flaquer J (2011) Méthodes probabilistes d'analyse de fiabilité dans la logique combinatoire. Theses, Télécom ParisTech. https://pastel.archives-ouvertes.fr/pastel-00678275. Accessed 5 May 2024
46. Wang Z, Zhang G, Ye J, Jiang J (2021) Reliability evaluation of approximate arithmetic circuits based on signal probability. Proceedings of 2021 IEEE International Test Conference in Asia (ITC-Asia). pp 1–6. https://doi.org/10.1109/ITC-Asia53059.2021.9808704
47. Xiao J, Shi Z, Yang X, Lou J (2022) BM-RCGL: benchmarking approach for localization of reliability-critical gates in combinational logic blocks. IEEE Trans Comput 71(5):1063–1076. https://doi.org/10.1109/TC.2021.3071253
48. Xiao J, Chen W, Lou J, Jiang J, Zhou Q (2022) Identifying reliability-critical primary inputs of combinational circuits based on the model of gate-sensitive attributes. IEEE Trans Comput Aided Des Integr Circuits Syst 41(11):4708–4720. https://doi.org/10.1109/TCAD.2022.3142194
49. Xiao J, Zhu W, Shen Q, Long H, Lou J (2022) A pruning and feedback strategy for locating reliability-critical gates in combinational circuits. IEEE Trans Reliab. https://doi.org/10.1109/TR.2022.3197787
50. Yu Z, Zhipeng J, Min W, Hudi P, Shuhua P, Changhong Y (2015) Optimize the PTM circuit calculation using deo algorithm. Proceedings of 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation. pp 757–759. https://doi.org/10.1109/ICMTMA.2015.187
51. Zhan S, Chen C (2022) An efficient method for sequential circuit reliability estimation. Proceedings of 2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS). pp 1–4. https://doi.org/10.1109/MWSCAS54063.2022.9859273
52. Ziegler JF, Muhlfeld HP, Montrose CJ, Curtis HW, O'Gorman TJ, Ross JM (1996) Accelerated testing for cosmic soft-error rate. IBM J Res Dev 40(1):51–72. https://doi.org/10.1147/rd.401.0051
53. Zimpeck A, Meinhardt C, Artola L, Reis R (2021) Reliability challenges in FinFETs. Springer, Cham, pp 29–63. https://doi.org/10.1007/978-3-030-68368-9_3

Esther Goudet is a Doctoral student in Computer Science at Télécom Paris, LTCI and STMicroelectronics, Technology and Design Platforms (TDP), in Crolles. She holds a master degree in Cybersecurity at the Université de Grenoble Alpes (UGA). She is interested in researching algorithm for estimating the fault propagation in combinatorial circuits.

Fabio Sureau is an internship student at STMicroelectronics, TDP, in Crolles. He his in third year of master's degree at the engineer school SupGalilée, Université Sorbonne Paris Nord, in Computer Science.

Paul Breuil was an intership student at STMicroelectronics, TDP, in Crolles from June to August 2023, during his second year as an engineering student. He is now in third year of engineering cycle at the École polytechnique, Palaiseau.

Luis Peña Treviño graduated a PhD Track master's degree at IP Paris Institute, in the Embedded Systems and Information Processing program, in 2023. He is now a PhD student focusing on low-power optimization at STMicroelectronics, TDP, in Crolles.

Lìrida Naviner is a Professor at Télécom Paris, LTCI. She is Chevalier des Palmes Académiques and IEEE Senior Member. She holds research on the scientific challenges of IoT for the electronic industry.

Jean-Marc Daveau received his PhD in Microelectronics from the Institut Polytechnique de Grenoble (INPG) in 1997. He currently works as senior front-end design engineer in STMicroelectronics in Crolles (France). His research interests include safety assessment, reliability evaluation and fault tolerance of complex digital systems and processors, including SoCs radiation testing and hardening.

Philippe Roche received the M.S. (1995) and Ph.D. (1999) in semiconductor physics. His primary activities are Single Event Effects and Total Ionizing Dose, as well as Ultra Low Voltage IPs, from sub-0.25µm technologies down to FinFET 30 Angstroms. He has been serving in conferences since 1997 as session chair and short course instructor. Philippe has coauthored +220 papers and filed +75 patents and 3 trade marks in radiation hardening. He was appointed Regional Fellow and Technical Director R&D in 2013, then elected by the ST Board as Corporate Fellow in 2020. After 5 years in a product organization designing ASICs, Philippe is back to ST Central R&D (FTM/TDP), in charge of new R&D explorations with a team of senior experts, also acting as Head of Labs & Ecosystems management, with LETI & CNRS as key partners.