Soft Error Rate Analysis in Combinational Logic

Martí Anglada Sánchez

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Advisor: Antonio González, Departament d'Arquitectura de Computadors
Co-Advisor: Ramon Canal, Departament d'Arquitectura de Computadors

This thesis is submitted for the degree of

Master in Innovation and Research in Informatics (MIRI-HPC)

July 2015

Abstract

Soft errors, energetic-particle induced single event data upsets, have emerged as a key reliability concern in sub-100 nanometre technologies. Low operating voltage, small node capacitance and high packing density are primarily responsible for the soft error susceptibility. State-of-the-art processors need to fulfill reliability standards and, therefore, processor designers need powerful tools to estimate the vulnerability of their architecture.

Historically, studies in soft error analysis have been focused on SRAM because it occupies the majority of the die area in system-on-chips and microprocessors. As a result, there have been great improvements in the efficient protection of memory cells but techniques for protecting combinational logic still remain at power-hungry levels. One of the reasons is the complexity of analyzing soft error in logic, which results in inefficient solutions for the problem of estimating reliability. This complexity in comparison to memories is consequence of less regularity of designs and less susceptibility of soft errors due to masking effects.

We develop a model that computes the probability that a strike at the output of a gate has an impact in any output by traversing the circuits backwards from the outputs and gaining information about the logical masking using signal probabilities. The model can easily be extended to take into account other masking effects such as electrical masking and latch-window masking. To properly estimate signal probabilities, we combine a widely-adopted algorithm from the literature called *Possibilistic algorithm* with fast, exhaustive simulation of small reconvergent fanout cones. The model is validated with fault injection simulation on a set of small circuits used in similar works from the literature and the ISCAS85 benchmarks.

Results show enormous speedup with respect to fault injection and errors of 4% on small circuits and circuits with few dependencies. Larger errors appear on some circuits and motivate further research.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter presents the problem of soft errors in computer architecture, motivates their study for reliable processors and lists the objectives of the thesis.

1.1 Soft errors from a computer architecture perspective

Recent trends in semiconductor technology scaling have brought tremendous improvement in performance and power consumption for microprocessors. Device scaling translates to a reduction in the feature size and voltage levels, aimed at having smaller devices that require less current to switch, and thus can be operated at higher frequencies. Today's dies consist of more than $1.5x10^9$ transistors, with gate lengths of 22nm [25]. However, scaling in sub-100 nm lithographies has brought attention to reliability issues that were previously not as much of a concern.

In particular, nanoscale integrated circuits (ICs) have become highly sensitive to disturbances caused by ionizing radiation. Radiation-induced transients in ICs are primarily caused by low-energy neutrons coming from cosmic rays and alpha particles coming from packaging materials [4]. These particles collide with the silicon nuclei, leading to a chain of secondary reactions that deposit a dense track of electron-hole pairs as they pass through a p-n junction and causing a current pulse at the node that collects the charge (Fig. 1.1). This phenomenon is referred to as single event transient (SET).

If a sufficient amount of charge is collected by the junction, the SET results in a fault by flipping the logic state at the associated node. When a fault caused by a bit flip is captured by a storage element such as a memory cell or a register, a single event upset (SEU) occurs. In contrast with effects such as hot-carrier injection (HCI) [8], negative-bias temperature instability (NBTI) [7] or electromigration [6] that cause permanent failures, SEU does not damage the device and, because of that, are called *soft errors*.

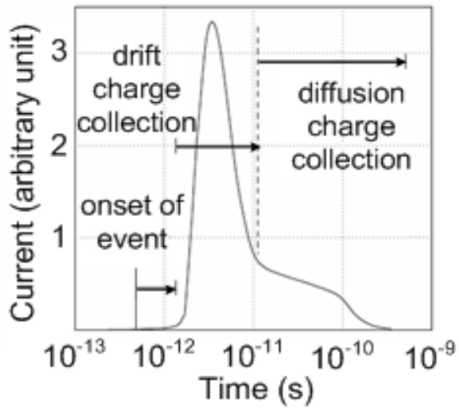


Fig. 1.1 Current pulse caused by a SET at the collection node

The rate at which soft errors occur is expressed in terms of failure in time (FIT), which measures the number of failures per 10^9 hours of operation. Typically, hard failure rates add up to 50-200 FIT, whereas the soft error rate (SER) can easily exceed 50000 FIT per chip [3]. As a consequence, the characterization and mitigation of soft errors in nanoscale ICs has become an important area of study.

1.2 Motivation

Traditionally, soft errors were tackled within the context of memory cells because a lot of the area of the chip is devoted to caches (Fig. 1.2). However, due to a continued reduction in both critical charge (the smallest charge that results in a soft error) and collection efficiency of transistors, SRAM SER has stayed constant over several generations. On the other hand, it was recently shown [38] that the SER in logic circuits has increased by five orders of magnitude in 8 technology nodes (from 600 nm to 50 nm) and is now comparable to the SER of unprotected memory.

Moreover, mechanisms such as error-correcting codes [32] or radiation-hardened latches [5] to protect memory elements from SET are well understood and have minimal overheads, so the probability of soft errors occurring in memory has lessened. In contrast, known techniques to reduce the SER in combinational logic such as triple modular redundancy [27] have substantial costs in term of performance and power consumption. Hence, it is important to develop accurate methodologies to analyze and quantify the impact of soft error in combinational logic in order to find efficient solutions to protect it. Furthermore, current designers must work toward a system-level FIT goal. There is, therefore, a need for analysis tools that can provide the designer the trade-off between the overheads of circuit mechanisms

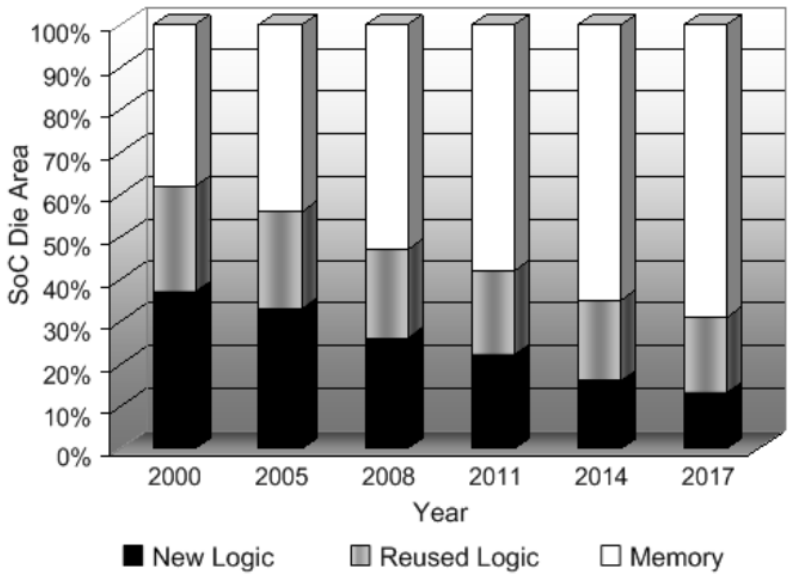


Fig. 1.2 Trend of memory and logic area on an SoC die [39]

for error resilience and the system-level FIT value of the circuit. These tools require not only accuracy but also speed, since they will be used several times through the design of the processor.

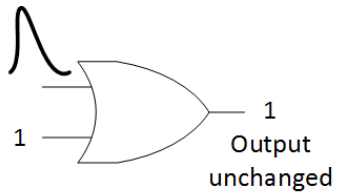
However, modeling and analyzing the SER in logic is more complex that in memory, since there are several masking effects that reduce the overall likelihood of the pulse being latched and causing an error:

1. Logical Masking
   Transient faults are blocked across the propagation path by a controlling value on the side-input of a gate. This effect can be seen in Fig. 1.3a: if a particle strikes at an input of an OR gate but one of the other inputs is in the controlling state (1), the strike will be completely masked and the output will not change.

2. Electrical Masking
   The pulse is attenuated (amplitude is reduced or rise and time fall times are increased) by the electrical properties of the gates throughout the logic chain and the resulting magnitude is insufficient to be latched (see Fig. 1.3b).

3. Latch-Window Masking
   Transient faults arrive at a state-holding element outside its latching time window (Fig. 1.4a) or when their pulse width is smaller than the setup time plus the hold time of the latching element (Fig. 1.4b).

(a) Logical masking



(b) Electrical masking

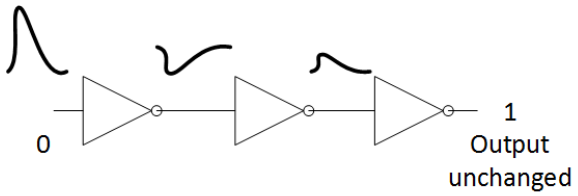
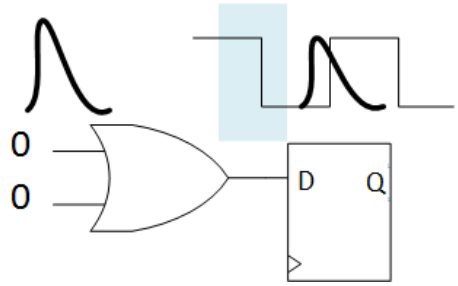


Fig. 1.3

(a) Masking due to the pulse timing



(b) Masking due to a short pulse

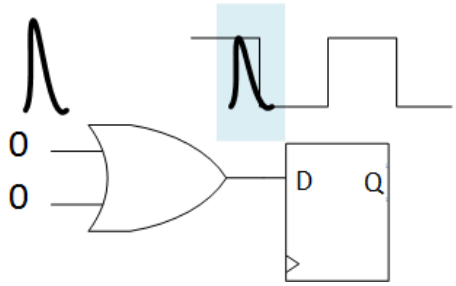


Fig. 1.4

The current trends in the improvement of technology, specially the increase of frequency, have led to an important reduction of the electrical and latch-window masking. Even though a significative amount of soft errors are masked due to these effects, the most important one nowadays is logical masking.

Reliability analysis is a nontrivial task due to the large size of ICs. In addition, it is known that the problem of determining whether the signal probability at a given node is nonzero is equivalent to the Boolean satisfiability (SAT) problem, a problem of determining if there exists an interpretation that satisfies a given Boolean formula. The SAT has been proven to be an NP-complete problem [16]. The problem of computing all signal probabilities in a circuit can be formulated as a random satisfiability problem, which is to determine the probability that a random assignment of variables will satisfy a given Boolean formula. The random satisfiability problem lies in a class of problems, called #P-complete, which is conjectured to be even harder than NP-complete.

This thesis proposes a methodology to evaluate the SER of a circuit, given its netlist and input distribution probabilities, considering the above masking effects. Most of the work that can be found in the literature (see Chapter 3) computes physical properties like critical charge and pulse propagation at the same time that the topology of the circuit is

taken into account to compute the effect of logical masking. These approaches are either too slow because they try to precisely model the pulses or too inaccurate because they want to be fast. Other authors have also used signal probabilities to evaluate reliability in logic circuits, but the algorithms used do not consider technology and only tackle logic masking. In comparison to those approaches, this work introduces additional parameters on top of the signal probabilities in order to accurately compute the effect of the technology on the circuit. Additionally, the pulse and pulse propagation modelling is decoupled from the algorithm that computes the vulnerability and stored in a library. This allows to have proper information about the technology behavior without the overhead of having to dynamically estimate it.

1.3 Objectives

The main objectives of this thesis are to develop a methodology to evaluate reliability in combinational circuits that:

- takes into account logical masking.
- can easily implement the effect of timing and electrical masking.
- is fast and accurate enough that can be used several times in the design of a microprocessor.
- receives as few inputs as possible so as to be integrated in more general design tools.

In order to achieve these requirements, a fault injector will be also developed to validate the accuracy and speed of the proposed method.

1.4 Organization

The thesis is organized as follows. Chapter 2 provides an overview of the physical mechanism of soft errors. Chapter 3 reviews the existing soft error modeling approaches for combinational logic. Chapter 4 describes the proposed modeling technique for reliablity and signal probability estimation. Chapter 5 details the methodology to validate the method. Chapter 6 shows the results of the validation, characterizes the weak spots of the model and how to improve it. Chapter 7 presents the conclusions of this dissertation and the future work.

Chapter 2

Radiaton-Induced Failures

This chapter reviews the dominant radiation sources, explains how soft errors are generated by the collection of radiation-induced charge and describes how latched errors can cause user-visible errors.

Single-event effects indicate any measurable or observable change in state or performance of a microelectronic component resulting from a single, energetic particle strike.

Soft errors are a particular subset of single-event effects that can affect digital circuits. A soft error occurs when radiation, directly or indirectly, induces a localized ionization capable of upsetting internal data states. The error is "soft" because the circuit or device itself is not permanently damaged by the radiation – if new data is written to the bit, the device will store it correctly. At a circuit level, a soft error corresponds to an erroneous output signal from a latch or memory cell:

1. A soft error in a memory element occurs when sufficient electrical charge is generated to invert the value stored in the memory element.

2. A soft error in logic occurs when the result of a malfunction due to radiation in a logic gate propagates to a storage element and is latched.

2.1 Radiation sources

2.1.1 Alpha particles

An alpha particle is a double-ionized helium atom ($^4He^{2+}$), composed of two protons and two neutrons. Packaging materials and solder contain traces of radioactive isotopes, which can produce alphas when they decay to a lower energy state. When an alpha particle travels

through a material, it loses its kinetic energy predominantly through interactions with the electrons of that material and thus leaves a trail of ionization in its wake. The emitted alphas have kinetic energies that are typically in the range of 4-9 MeV. For each electron-hole pair that is created, on average 3.6 eV of kinetic energy is lost [19]. This implies that an alpha can cause a burst of typically a million electron-hole pairs. As the alpha is losing kinetic energy, its speed is lowered, which increases the available time to induce electron-hole pairs. Therefore, the charge generation rate increases with the distance traveled by the alpha and has a maximum near the end of the trajectory. The range of an alpha particle is a function of its energy and the properties of the material in which is traveling. In silicon, the range for a 10-MeV alpha particle is less than 100 μm. Thus, alpha particles from outside the packaged device are clearly not a concern–only alpha particles emitted by the device materials and packaging materials need to be considered. Using Ultra-Low-Alpha prouduction materials, manufacturers such as TSMC, ST or TI claim that alpha particle emission rates of packaged devices nowadays are 0.001 α/cm^2h, causing a 20% of the soft errors [30].

2.1.2 High-energy cosmic neutrons

Another cause of soft errors is cosmic rays, high-energy particles from outer space. When they arrive at Earth, they collide with the nuclei of atoms in the upper atmosphere, creating cascades of second and higher generation particles, as seen in Fig. 2.1.

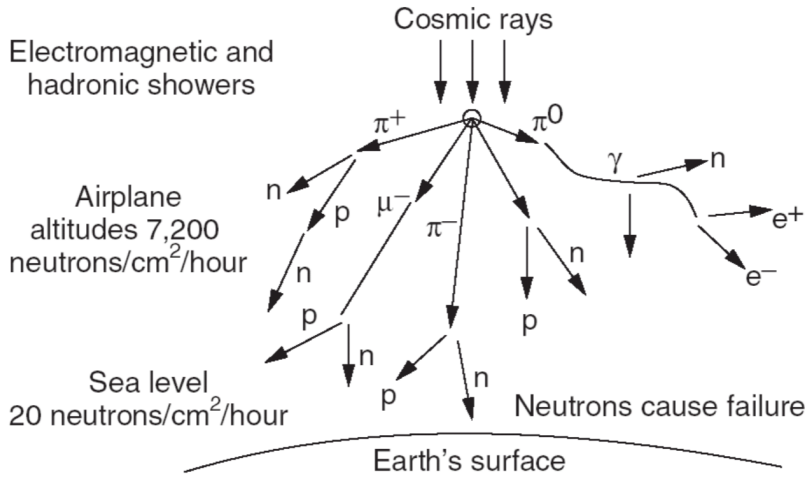


Fig. 2.1 Cosmic ray desintegration, causing a cascade of nuclear reactions

Most charges from cosmic rays are trapped or deflected by the geo-magnetic field of the Earth, and only 1% of the primary flux reaches the sea level [45]. At terrestrial altitudes, the predominant particles with strong nuclear force are protons, neutrons, electrons,

muons, and pions. Muons and pions are short-lived and protons and electrons are attenuated by Coulombic interactions with the atmosphere. Therefore, neutrons are the dominant cosmic particles producing soft errors, because of their relatively high flux and their stability. The SER of devices due to cosmic rays can be estimated with sufficient accuracy by just considering neutron interactions. Since neutrons are uncharged particles, they do not directly generate ionization in silicon, and its flux alone does not define the cosmic component of SER. Neutrons interact with chip materials elastically and inelastically. Inelastic reactions typically end with the excited nucleus breaking into lighter fragments.

When the silicon nucleus fragments in these inelastic reactions, the resultant products are a lighter ion with additional particles (neutrons, protons, and/or alpha particles). Kinetic energy is shared among the particles and momentum is conserved, so the particles tend to be emitted in opposing directions. As the energy of the incident neutron gets higher, the number of reaction pathways increases. Because neutrons create ionizing particles indirectly, the tracks of the created secondary ions can start anywhere and in any direction in the device, as seen in Fig. 2.2. This is in contrast with alpha particles or other light charged particles with sufficient ionizing power, which arrive from outside the device. This implies that in the case of neutrons, the number of geometrical situations with respect to ion tracks and sensitive regions of the devices is much larger. Furthermore, a collision of a neutron with a silicon nucleus results in more than one ionizing particle. Also, unlike with alpha particles, most of the neutrons entering a device pass through the silicon without interaction, and only a small fraction gets involved in a nuclear reaction.

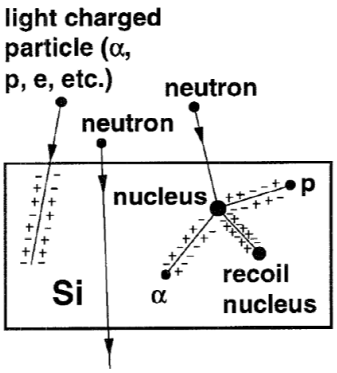


Fig. 2.2 Generation of electron-hole pairs in silicon by high-energy neutrons

In contrast with alpha particles, the cosmic neutron flux cannot be reduced significantly at the chip level with high purity materials or shielding, since materials such as concrete (that reduces cosmic radiation at a rate of approximately 3x per meter) cannot be applied

in personal desktops or portable devices. Cosmic ray SER must therefore be dealt with by reducing device sensitivity, either by design or process modifications.

2.2 Physical mechanism

After an ionizing particle strikes a semiconductor device, it creates a highly conductive, neutral and localized column of electron-hole pairs. When the particle-induced charge track traverses or comes close to a reverse-biased p-n junction, electron-hole pairs are rapidly collected due to the high electric field of the depletion region of the junction [12]. The created electrons drift to the higher potential of the n-diffusion while holes drift to the lower potential of the p-substrate. The equipotential lines of the depletion layer rapidly spread down and envelope the entire length of the track, as shown in Fig. 2.3. The electric field spreads down along the track into regions that originally were field-free, distorting the depletion region into a funnel shape. Within tens of picoseconds, the generated carrier density near the junction is comparable to the substrate doping, and the disturbed field relaxes back to its original position. Charge collection through diffusion [13] continues for several nanoseconds: while the excess electrons and holes deeper in the substrate diffuse away, part of the electrons are captured by the sensitive node.

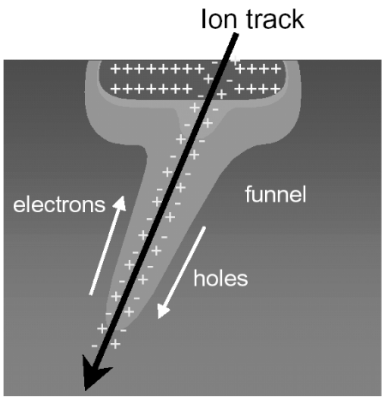


Fig. 2.3 Charge generation and collection after a particle hit

In general, the farther away from the junction that the particle strikes, the smaller the amount of charge that will be collected, and thus the less likely it is that the event will cause a soft error. The magnitude of the collected charge (Q_{coll}) depends on a complex combination of factors including the size of the device, substrate doping, the type of particle, its energy and trajectory and the state of the device. Additionally, since in integrated circuits a node is in close proximity with other nodes, charge sharing and parasitic bipolar action [2] can

greatly influence the amount of charge collected. Q_{coll} does not result in a soft error unless it exceeds the minimum charge required to cause a change in the data stated, named critical charge (Q_{crit}) [11]. Q_{crit} is difficult to model because it is a function of the node capacitance, operating voltage and strength of the restoring mechanisms connected to the node.

Chapter 3

Related work

This chapter covers the literature for soft error analysis and estimation in combinational logic and the related work to this thesis' approach. Simulation techniques and analytical tools and models are reviewed, highlighting the trade-offs between accuracy and execution time and the means to cope with the different masking mechanisms.

3.1 Fault-injection-based approaches

The industry standard tool for circuit simulation is HSPICE [40]. HSPICE can be used to estimate the SER of a circuit by computing the critical charge (Q_{crit}) as a measure of the circuit susceptibility to particle strikes. For a soft error to occur at a specific node in a circuit, the collected charge at that particular node should be more than Q_{crit}. This value is obtained from the circuit by injecting a series of current pulse and noting the minimum which upsets the device. Since HSPICE is a device-level tool which models technology nodes, very precise results can be achieved at the expense of long simulations. In practice, execution times are infeasible for circuits exceeding a few gates and, as a result, is only used as a technique to compare the accuracy of faster methods [38].

As in other areas of computing where analytical solutions are complicated to solve, Monte Carlo methods are often used in reliability analysis. Monte Carlo is a widely known simulation-based approach where experimental data are collected to characterize the behavior of a circuit by randomly sampling its activity [20]. In the context of reliability estimation, the samples correspond to a series of pulses to evaluate the behavior of transistors or to a series of bit flips to evaluate the behavior of logic gates. Although not all the space has to be explored, the big drawback is that numerous pseudorandom numbers need to be generated, and a large number of simulation runs must be executed to reach a stable result. In process of achieving

relatively stable results, certain statistical parameters such as coefficient of variance (σ/μ) are usually used as the stopping criteria. According to [17], the number of MC runs varies around 10^5-10^7 with acceptable coefficients of variance of 0.001 and it is empirically shown that the result becomes relatively stable after 10^4 runs.

In [28] a tool based on Monte Carlo and integrated with the IBM chip design process is presented. The program requires a number of different inputs (semiconductor description, impurity concentration and depth, alpha particle pattern, wiring rules, critical charge at the sensitive terminals, chip dimensions, geographic coordinates, number of bits in the chip) to perform very accurate simulations of the effect of particle strikes. Alpha particle emission from the chip is computed and Monte Carlo simulations are run to obtain different radiation emissions. With them, nuclear collisions are simulated and energies and directions of particles generated. In an iterative process, the carriers are moved through the transistor area and the new coordinates are used to determine whether a junction has crossed and with which charge, so as to determine if the bit is flipped. The SER of the whole chip without taking into account masking effects can be very accurately computed at the cost of large execution times.

Zhang *et al.* developed a methodology [44] based on graph theory, conditional probability and fault simulation. The circuit is decomposed into a collection of nodes with a gate between each pair of nodes, and each gate is modeled as an equivalent inverter. The circuit is simulated to obtain a series of conditional probabilities (charge collection by the p-type or n-type drain, collection by a node or by a flip-flop) to compute the likelihood of a pulse being latched. Logical masking is emulated in the inverter chain by using logic functions in adjacent nodes. A path-search algorithm is used to find all paths between primary outputs and internal circuit nodes. The SER computation is repeated for a number of input vectors. Using inverter chains may lead to inaccuracies in electrical and timing masking, and only using a subset of input vectors can not show the variations for different input states. The results show great speedup in execution time at the expense of only a 4% loss in accuracy, but the authors only provide results for circuits of the order of 10 gates.

The approach in [18] finds what set of latches are affected by which logic gates, timesteps, magnitudes of collected charges and input vectors. The circuit is first simulated to find the logic states of all the gates. Then, the FIT rate of combinational nodes is estimated by determining which charges and times of strikes can lead to an event of failure. This is achieved by pre-characterizing the magnitudes of collected charges required to cause a flip

for each of the gates and finding sensitized paths from the gates to latches. Finally, an additional step using statistical fault injection is used to determine the probability that a system-level specification fails due to a strike in a combinational node. Their approach is much faster at running thousands of input vectors than other techniques because the fault injection is less present, but also needs an exponential number of runs to traverse repeatedly the input space. The results presented only show great gains in execution time and not accuracy for large ISCAS benchmark circuits.

3.2 Pulse-based approaches

Another set of techniques try to reduce execution time by not having to simulate the behavior of the gates at runtime: instead, the response of the gates to the pulses is modelled. In [43], the methodology presented uses a previously-characterized technology library to emulate the effects of pulse generation and propagation. The error pulses are encoded and propagated using binary decision diagrams (BDD) [9]. A strike modifies the BDD adding extra information, such as the arrival time, pulse and voltage of the pulse. Logical masking is captured by propagating the error only if the path from the site of the fault is sensitizable under the assignment of the primary inputs. Determining the output of the BBD is done through logic operation and look-up tables. The results present a great speedup in time (in the order of minutes) with an average error of 12% in small circuits.

Rajamaran *et al.* also proposed in the same year to capture current-voltage characteristics beforehand via HSPICE simulations and compile their results into a set of mathematical equations that would provide accurate and quick information at dynamically. In their work, more data, such as gate delays and time window for flip-flops is precomputed so as to lose the least amount of precision possible. Time and accuracy results are given for small circuits: with only one minute of execution, they achieve an average error of 6.5%. For bigger ISCAS benchmark circuits, only time results are provided, and an average execution time of 6 hours is reported.

The work presented in [10] is based on decomposing transient faults into two transitions for analysis: rising edge and falling edge. Each edge is processed using an analytical approach and statistical static timing analysis. By expressing timing quantities in closed form, the arrival time and required arrival time can be propagated through a timing graph using a linear-time block-based statistical timing algorithm. The SER of every gate is formulated by integrating the products of particle-hit rate and the error probability over the range of

charges. The error probability depends on the masking effects: electrical and timing masking (computed with first order forms for the hit and propagation pulses) and logical masking (computed with signal probabilities). Although this approach is not as fast as others, it still achieves more than 250x speedup in relation with Monte Carlo with only 3% error and can take into account process variations.

In order to reduce the computing effort to compute electrical masking, the authors in [41] present a model based on two lookup tables per transistor: a drain current model and a capacitance model. The motivation behind is that without accurately modeling large currents at the linear region or the sensitivity of effective capacitances, an analytical model cannot match SPICE simulations. With these two models, they can analyze combinational logic considering electrical masking with a great speedup with respect to simulations using fault injection. They provide results for an inverter chain and a two-input NAND gate chain of lengths up to 6 and report errors of less than 6%.

The proposal introduced in [22] to avoid large computation times also relies in information known statically. The authors bypass modeling technology properties by choosing adequate critical charge values for each technology node. They first use the SER model for SRAM circuits to account for the SER in logic without any masking, and then they use one reducing factor for masking property. For electrical masking, they assume that pulses with a certain charge will be able to propagate to any number of gates, and they extract that charge using information of the particle flow and the technology node. For timing masking, they compute the probability of a pulse with a certain charge has enough duration to be latched. Finally, they set a value of 0.5 for the logical masking reducing factor based on the mean of several empirical results. Since the values of these factors are known independently of the size of the circuit, it is the fastest approach in the literature, requiring constant time. With regards to accuracy, the model outputs a FIT rate in the same order of magnitude than much more complex models if the few parameters used (collection efficiency, critical charge, latching window, pulse duration and logical masking ratio for the input vector) are correct.

The most current soft error analysis technique was presented by Kwon *et al.* [26]. In their work, the SER of a gate is computed by taking into account all paths from other gates to it and multiplying the pulse generation probability by the probability of the masking factors. The logic gate traversal starts from the nearest gates to the outputs and analyzes each gate backwards from them. In order to calculate the logical propagation probabilities without using static probability, each gate contains two types of look-up tables (an attenuated

pulse with table that accumulates pulse sums and a validation table to compare technological parameters) and node conditional probability. The authors provide results for all ISCAS85 benchmark circuits, with execution times of less than an hour and an average error of less than 16%.

3.3 Probability-based approaches

The work presented in [14] uses signal probability as an indication of signal reliability. In the case of a fault-prone logic signal, four possible states can be defined: correct 0, correct 1, incorrect 0 and incorrect 1. This is represented in a 2x2 matrix. The reliability of a gate is represented by two possible states: correct operation and failure in a probabilistic transfer matrix (PTM). The output probabilities are computed by multiplying the input probabilities and the gate's PTM. The signal probability of the output node in a circuit corresponds to the signal reliability of the circuit, and it is computed by propagating the input signal probabilities through the gates of the circuit in topological order. With a straightforward signal probability computation algorithm, the methodology allows the evaluation of the logical masking capability of the studied circuit. The process of combining gate probability matrices implicitly takes into account the single dependency between gates by considering the underlying joint and conditional probabilities and, as a result, the calculation is exact. However, the scalability is limited because it has an exponential space requirement for the matrices. The results that the authors provide, while accurate, are only shown for circuits with sizes around 50 gates.

Another way to use signal probability in reliability computation is with the probabilistic gate model (PGM) [15]. Input signals of each gate are assumed to be independent. Under this assumption, the output probability of each gate can be easily calculated using the information of input signal probabilities and gate error rate. The output probability can be used recursively as the input information at next level of gates. Circuit reliability is analyzed by exhaustively evaluating each input combination. The obvious disadvantages are the impossibility to enumerate all input combinations when the number of inputs is high and the inaccuracy caused by signal independence assumption. Even if reconvergent fanouts are studied independently to remove inaccuracies, since the number of fanouts is comparable to the number of gates in the circuit, the complexity is still an exponential function of the circuit size, making it infeasible in general for large circuits.

The idea of PGM was adopted in [23] as a part of a two-pass algorithm. It starts from the output and constructs error probability equations pushing the proper susceptibility gate tables onto a stack. When the primary inputs are reached, another pass is applied to solve the equations by using input vectors. This is repeated exhaustively for all input patterns. The model does not take into account electrical or time masking, and accuracy results, while good, are only provided for circuits of the size of a 2-bit full adder.

In order to achieve great scalability, Pagliarini *et al.* suggested a linear-complexity solution in [31]. Each gate has associated the number of faults it generates and a finite-state machine with the following states: waiting for the computation of the previous' gates faults, ready to send computation to the following gates and several that represent a failure in one or more inputs and compute the generated faults accordingly. When a gate has all its predecessors ready, it switches to the state corresponding to the number of faulty inputs and computes the output number with quick, empirically-determined, gate-dependent shift right operations. Even though this approach does not present great accuracy numbers and only takes into account logical masking, it is fully synthesizable on an FPGA, which implies extraordinary speed. Each transition in a finite-state machine can be done in a single clock cycle and all gates that have their operands ready can operate in parallel.

A different manner of using signal probability in the reliability-estimation problem is via Bayesian networks (BN) [21]. A BN is a form of graphical model which is having fundamental components, nodes and links. These links combine and make directed acyclic graphs by joining several nodes. The directed acyclic direction lines are representing the joint probabilities among the nodes. Every node has its own conditional probability table according to its gate type. In [34], the authors use BN to evaluate the reliability of the circuit by comparing a circuit encoded as error-free with a circuit encoded with errors. Even though the time complexity is exponential, it is much smaller than PTM, since is proportional to the largest clique[1] in the graph in comparison with the number of inputs and gates. When comparing to fault simulation estimates of the order of 1000 samples in ISCAS85 benchmarks, results show errors of less than 7% with execution times of less than a minute.

The methodology presented in this thesis is also based in signal probabilities, eliminating the use of fault injection or pulse modeling at runtime. Unlike other probability-based approaches, it does not rely on complex mathematical models to evaluate masking effects.

[1] A subgraph in which every pair of distinct nodes are adjacent

Instead, it uses basic Boolean algebra notions and a simple graph traversal technique from the outputs of the circuit to the primary inputs to incrementally gain information about the logical masking properties of the circuit. The graph is traversed in such a way that technology information can be added as an input of the algorithm in order to compute electrical and latch-window masking. The straightforwardness of the implementation and the simplicity to obtain its inputs make our methodology an ideal candidate to be the reliability estimation tool within a bigger design program.

Chapter 4

Methodology to evaluate masking effects

This chapter presents the details of a comprehensive model for the analysis of the soft error rate in combinational logic as well as an example explaining how the model works. Reconvergent fanouts and their importance in signal probability are addressed.

4.1 Description of the algorithm

The proposed algorithm takes three inputs: the netlist of a combinational circuit, a set of input probabilities and a set of transition probabilities. The netlist of the circuit is a list of gates and their connectivity, which corresponds to a directed acyclic graph defined by the union of the list of gates G (vertices), the set of connections C (directed edges), the set I of inputs of the circuit and the set Ω of outputs of the circuit.

- Each element of G is formed by a pair $<\varphi, k>$, where φ is a Boolean operation (such as NOT or NAND) and k is the number of inputs of the gate.

- Each element of C is a connection among one output of a gate and one or more inputs of succeeding gates. This is formally defined as a pair $<g_i, \Gamma_i>$, where g_i is the ith gate of G and Γ_i is a set of gates $\{g_j^a, g_k^b, g_l^c, \dots\}$, where subscripts j,k and l indicate a particular gate from G and superscripts a, b and c indicate the input (0, 1, $\dots$, k-1) of the gate that is connected to the output of g_i.

- Each element of I is a pair $<i, m>$, where i designates the ith gate of G and m corresponds to input number m ($0 \leq m < k$) from g_i.

- Each element of Ω is an index i, which indicates that gate g_i produces an output of the circuit.

Every element m from I has associated an input probability, which corresponds to the probability that the value of m is 0. Every element g_i of G has associated two transition probabilities $T_i^{out}[0 \rightarrow 1]$ and $T_i^{out}[1 \rightarrow 0]$ which correspond to the probability that, given a strike, the output at g_i transitions, respectively, from 0 to 1 and from 1 to 0. The algorithm computes a set of vulnerability probabilities V which indicate, for every connection in C, the probability that a strike in that connection will be propagated to one or more outputs in Ω. Finally, it computes the vulnerability of the whole circuit based on the above probabilities. In order to compute V, the graph is traversed starting at the outputs and visiting a gate only when all its successors have already been visited.

To compute the probability that a strike in the connection at the output of a gate g_i is propagated, the probability that the strike is not masked by a succeeding gate is computed. To do that, all the paths from this connection to any output of the circuit are taken into account. A pseudo-code summarizing the procedure is shown below:

Algorithm 1 Algorithm to estimate the SER in a circuit

```
1:  Compute input probabilities of all gates
2:  Inititialize Queue with output gates
3:  while Queue is not empty do
4:      gate ← first element of Queue
5:      mark gate as visited
6:      for each parent ← {Gates in Predecessors(gate)} do
7:          if all gates in Successors(parent) have been visited then
8:              Queue ← Queue ∪ parent
9:          end if
10:     end for
11:     vulnerability(connection(gate)) = 1
12:     for each j ← {Inputs in Successors(gate)} do
13:         Prob_j ← Compute vulnerability of any path from gate to a circuit output that
        includes the connection from gate to input j
14:     end for
15:     Compute vulnerability of the connection at the output of gate with all Prob values
16: end while
17: Compute the vulnerability of the circuit based on the vulnerabilities of all connections
```

The first step of the algorithm (Line 1) computes the input probabilities of all the gates, based on the input probabilities of the circuit. This is done by traversing all the gates of the circuit from inputs to outputs and taking into account the function performed by every gate. For instance, the probability P_{NAND} that at the output of a 2-input NAND gate is a 0 knowing

the probability P_i that there is a 0 at the input i is:

$$P_{NAND} = (1 - P_0) * (1 - P_1) \tag{4.1}$$

which is the probability that both inputs are 1.

The probability P_{AND} that at the output of a 2-input AND gate is a 0 knowing the probability P_i that there is a 0 at the input i is:

$$P_{AND} = 1 - ((1 - P_0) * (1 - P_1)) \tag{4.2}$$

which is the probability that at least one input is 0.

The probability P_{OR} that at the output of a 2-input OR gate is a 0 knowing the probability P_i that there is a 0 at the input i is:

$$P_{OR} = P_0 * P_1 \tag{4.3}$$

which is the probability that both inputs are 0.

The algorithm uses a queue, created at Line 2, to store in the proper order the gates whose connections at the output are ready to be visited. The queue is initialized to the gates that produce the outputs of the circuit to ensure a traversal that visits all gates. The main loop occurs in Lines 3 through 16: not until every gate has been visited (equivalent to the queue not being empty), a gate is removed from the queue (Line 4) and it is marked as visited (Line 5). A new gate is enqueed to be examined later (Line 8) only if all their successors have been visited (Line 7).

Output connections of the circuit do not logically mask any SET, so they have a vulnerability of 1. Every connection is set with this initial vulnerability (Line 11), and then is changed according to its successors, as described below. In Line 12, every input from all the successor gates of the gate being visited are considered. In line 15, we compute the vulnerability of the connection at the output of the gate for all alternative paths to the output of the circuit. For this purpose, we first compute the individual vulnerability for each set of paths that include the connection from the output of the gate to a particular input of a successor gate (Line 13). Then we compute the vulnerability of the connection at the output of the gate as a combination of the individual vulnerabilities, assuming that the probability of masking is independent for different paths.

For instance, in the circuit of Figure 4.1, when we compute the vulnerability of the connection at the output of gate 4, we need to first compute the vulnerability of all paths going from gate 4 to the output of the circuit through input 0 of gate 1, and then the same for input 0 of gate 2. Finally, we compute the vulnerability of the connection by combining these two probabilities assuming that masking in each of them is independent.

The logical masking of each individual gate is computed by taking into account the function that it performs and the previously computed input probabilities for the gate. As an example of this, we use a 2-input NAND gate:

1. If both inputs are 0, a particle strike at any input does not affect the output.
2. If one input is 1 and the other is 0:
   (a) If the input set to 1 suffers a transition from 1 to 0, the output is not affected.
   (b) If the input set to 0 suffers a transition from 0 to 1, the output changes.
3. If both inputs are 1, a transition from 1 to 0 at any input changes the output.

This is summarized in Table 4.1:

Table 4.1 Logical masking of a 2-input NAND gate

Input 0	Input 1	NAND Output	Output if Input 1 changes	Output if Input 2 changes
0	0	1	1	1
0	1	1	0	1
1	0	1	1	0
1	1	0	1	1

So, the effect of not logically masking a SET for a 2-input NAND gate is:

$$NotMasking_{NAND2} =$$
$$P_0 * (1 - P_1) * T_0^{in}[0 \rightarrow 1] \quad (4.4a)$$
$$+ (1 - P_0) * P_1 * T_1^{in}[0 \rightarrow 1] \quad (4.4b)$$
$$+ (1 - P_0) * (1 - P_1) * (T_0^{in}[1 \rightarrow 0] + T_1^{in}[1 \rightarrow 0]) \quad (4.4c)$$

where 4.4a and 4.4b correspond to the cases from 2b and 4.4c corresponds to the cases from 3. $T_j^{in}[n \rightarrow m]$ indicates the probability of a transition from n to m at input j, which corresponds to $T_i^{out}[n \rightarrow m]$ from the input specification, where i is the gate whose output is connected to input j.

In a similar fashion, we can formulate this effect for any gate, for instance:

Table 4.2 Logical masking of a 2-input OR gate

Input 0	Input 1	OR Output	Output if Input 1 changes	Output if Input 2 changes
0	0	0	1	1
0	1	1	1	0
1	0	1	0	1
1	1	1	1	1

$$NotMasking_{OR2} = P_0 * P_1 * (T_0^{in}[0 \to 1] + T_1^{in}[0 \to 1]) + P_0 * (1 - P_1) * T_1^{in}[1 \to 0] + (1 - P_0) * P_1 T_0^{in}[1 \to 0] \quad (4.5)$$

Table 4.3 Logical masking of a NOT gate

Input	NOT Output	Output if Input changes
0	1	0
1	0	1

$$NotMasking_{NOT} = P_0 * T_0^{in}[0 \to 1] + (1 - P_0) * T_0^{in}[1 \to 0] \quad (4.6)$$

Table 4.4 Logical masking of a 2-input AND gate

Input 0	Input 1	AND Output	Output if Input 1 changes	Output if Input 2 changes
0	0	0	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

$$NotMasking_{AND2} = P_0 * (1 - P_1) * T_0^{in}[0 \to 1] + (1 - P_0) * P_1 * T_1^{in}[0 \to 1] + (1 - P_0) * (1 - P_1) * (T_0^{in}[1 \to 0] + T_1^{in}[1 \to 0]) \quad (4.7)$$

Finally, in line 17, we compute the vulnerability of the whole circuit by combining the vulnerabilities of all its connections. One way to do that is to assume that the strikes are equiprobable in all connections. In this case, the vulnerability of the circuit is the arithmetic mean of the vulnerabilities of all connections.

4.1.1 Running example

Figure 4.1 shows an example circuit to illustrate how the algorithm works.

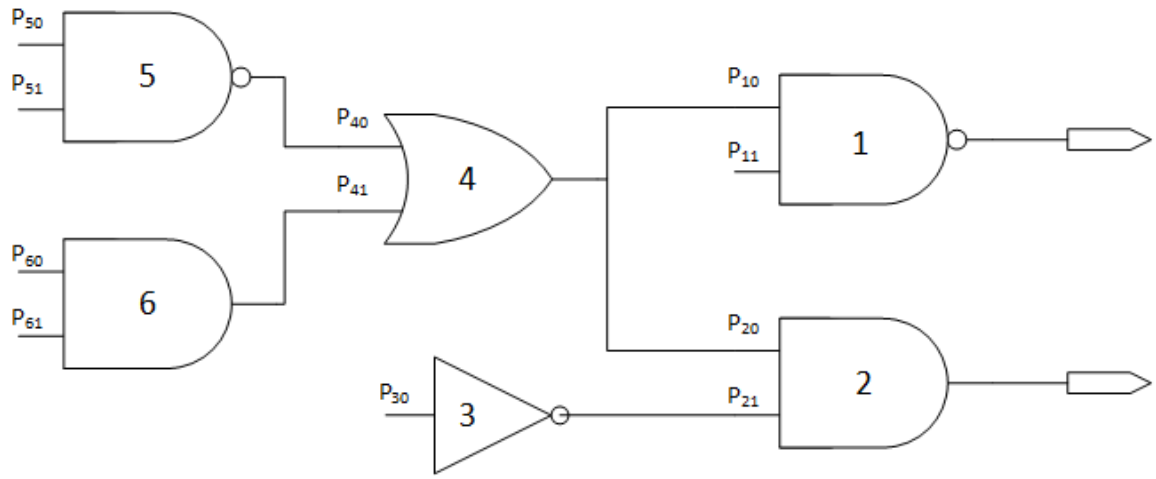


Fig. 4.1 Example circuit

An arbitrary input probability distribution of the circuit is listed in Table 4.5. In actual processor design, these input distributions would be known by profiling on realistic setups.

Table 4.5 Example input probability distribution

Input name	Probability of being 0
11	0.1
30	0.3
50	0.7
51	0.5
60	0.1
61	0.25

The probabilities that a strike causes a change in each connection are also needed. For this example, we assume the probabilities listed in Table 4.6. The authentic probabilities are obtained via SPICE simulations on the gates, testing for which inputs and which terminals the gate is sensitive to a bit flip.

Table 4.6 Example transition probability distribution

Input name	T[0→1]	T[1→0]
10	0.3	0.8
20	0.3	0.8
21	0.7	0.5
40	0.8	0.3
41	0.9	0.4

Figure 4.2 illustrates the first step of the algorithm, after which all the nodes have their input probabilities computed using equations such as (4.1), (4.2) and (4.3).

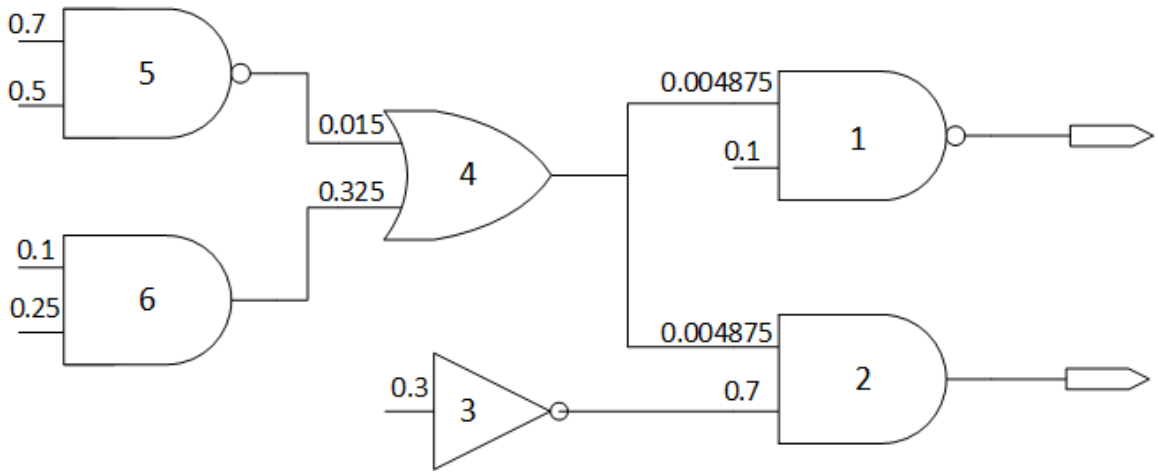


Fig. 4.2 Circuit with input probabilities propagated

After the propagation of the input probabilities, the queue is initialized with the output gates of the circuit:

Gates in Queue	1,2
Current Gate	

And the main loop begins.

Iteration 1

Gates in Queue	2
Current Gate	1

Gate 1 is popped out from the queue and it is marked as visited. Its predecessor (Gate 3) has two successors: Gate 1 and Gate 2. Gate 1 is being visited, but Gate 2 has not been visited yet. Therefore, since all and its successors have not been visited, Gate 3 is not yet added to the queue. The vulnerability at the output of Gate 1 is set to 1 and, since it does not have any successors, it remains 1.

$$Vulnerability(Output_{Gate1}) = 1$$

Iteration 2

Gates in Queue	3,4
Current Gate	2

Gate 2 is popped out from the queue and it is marked as visited. Its predecessors (Gate 3 and Gate 3) have all of their successors visited, so they are added to the queue. The vulnerability at the output of Gate 2 is set to 1 and, since it does not have any successors, it remains 1.

$$Vulnerability(Output_{Gate2}) = 1$$

Iteration 3

Gates in Queue	4,5,6
Current Gate	3

Gate 3 is popped out from the queue and it is marked as visited. Its predecessors (Gates 5 and 6) are added to the queue because all of their successors have been visited. The vulnerability at the output connection of the gate is first set to 1, but then it will be updated because it has two successors. Its first successor is a NAND gate, so its effect on the logical masking is given by equations (4.4a - 4.4c). Its second successor is an AND gate, so its effect on the logical masking is given by equation (4.7).

Both the input probabilities and the probability that the strike switches the proper input to a particular value (i.e., the $T_0^{in}[0 \to 1]$, $T_0^{in}[1 \to 0]$, $T_1^{in}[0 \to 1]$ and $T_1^{in}[1 \to 0]$ equations (4.4a - 4.4c) and (4.7)) are known, since they are inputs to this algorithm. With this information, the vulnerability of each successor can be calculated:

$$\begin{aligned} NotMasking_{NAND2} &= P_0 * (1 - P_1) * T_0^{in}[0 \to 1] + (1 - P_0) * (1 - P_1) * T_0^{in}[1 \to 0] \\ &= 0.004875 * 0.9 * 0.3 + 0.995125 * 0.9 * 0.8 = 0.718 \end{aligned}$$

$$\begin{aligned} NotMasking_{AND2} &= P_0 * (1 - P_1) * T_0^{in}[0 \to 1] + (1 - P_0) * (1 - P_1) * T_0^{in}[1 \to 0] \\ &= 0.004875 * 0.3 * 0.3 + 0.995125 * 0.3 * 0.8 = 0.2 \end{aligned}$$

Once the probability of each successor is calculated, the vulnerability at the output of the current gate can be computed.

$$\begin{aligned} Vulnerability(Output_{Gate3}) = {} & NotMasking_{NAND2} * Vulnerability(Output_{Gate1}) \\ & + NotMasking_{AND2} * Vulnerability(Output_{Gate2}) \\ & - [NotMasking_{NAND2} * Vulnerability(Output_{Gate1}) \\ & * NotMasking_{AND2} * Vulnerability(Output_{Gate2})] = 0.78532 \end{aligned}$$

Iteration 4

Gates in Queue	5,6
Current Gate	4

Gate 4 is popped out from the queue and it is marked as visited. Since it has no predecessors, nothing is added to the queue. The vulnerability at the output connection of the gate is first set to 1, but then it will be updated because it has a successor. Its successor is an AND gate, whose effect on the logical masking is given by equation (4.7).

$$NotMasking_{AND2} = 0.995125 * 0.7 * 0.7 + 0.995125 * 0.3 * 0.5 = 0.63688$$

Then, the vulnerability at the output of the gate can be computed.

$$Vulnerability(Output_{Gate4}) = NotMasking_{AND2} * Vulnerability(Output_{Gate2}) = 0.63688$$

Iteration 5

Gates in Queue	6
Current Gate	5

Gate 5 is popped out from the queue and it is marked as visited. Since it has no predecessors, nothing is added to the queue. The vulnerability at the output connection of the gate is first set to 1, but then it will be updated because it has a successor. Its successor is an OR gate, whose effect on the logical masking is given by equation (4.5).

$$NotMasking_{OR2} = 0.015 * 0.325 * 0.8 + 0.985 * 0.325 * 0.3 = 0.0999375$$

Then, the vulnerability at the output of the gate can be computed.

$$Vulnerability(Output_{Gate5}) = NotMasking_{OR2} * Vulnerability(Output_{Gate3}) = 0.078484$$

Iteration 6

Gates in Queue	
Current Gate	6

Gate 6 is popped out from the queue and it is marked as visited. Since it has no predecessors, nothing is added to the queue. The vulnerability at the output connection of the gate is first set to 1, but then it will be updated because it has a successor. Its successor is an OR gate, whose effect on the logical masking is given by equation (4.5).

$$NotMasking_{OR2} = 0.015 * 0.325 * 0.9 + 0.015 * 0.675 * 0.4 = 0.008435 \tag{4.8}$$

Then, the vulnerability at the output of the gate can be computed.

$$Vulnerability(Output_{Gate5}) = NotMasking_{OR2} * Vulnerability(Output_{Gate3}) = 0.006626 \tag{4.9}$$

After these six iterations, the algorithm terminates and every connection stores the probability that, if there is a SET in it, the effect of the strike will be visible in one or more

outputs. Finally, the vulnerability of the circuit is computed as the arithmetic mean of all these probabilities. Figure 4.3 shows the results of the algorithm.

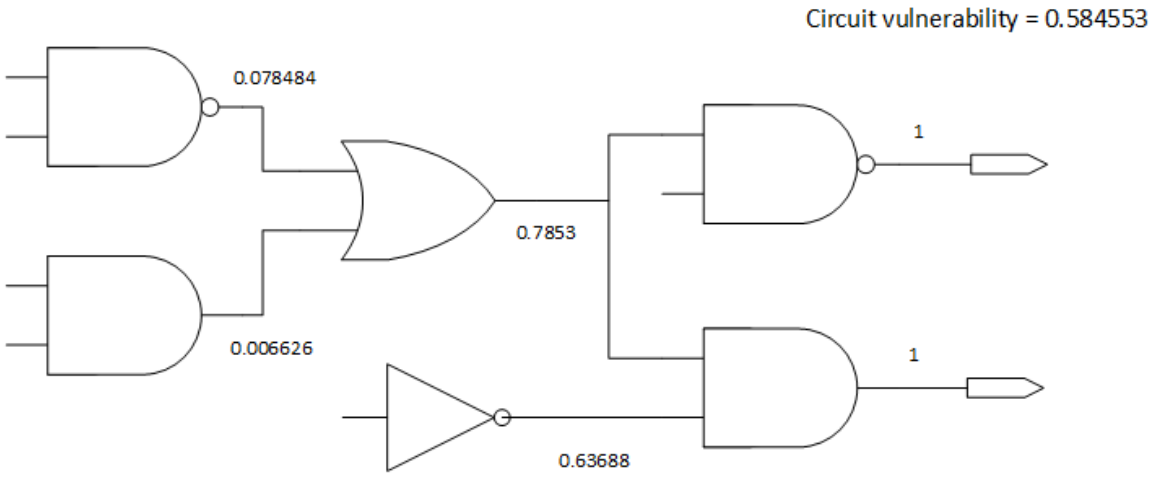


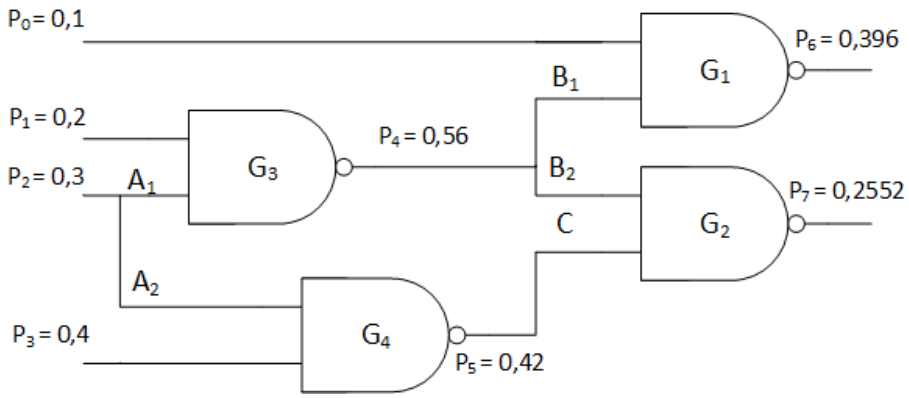
Fig. 4.3 Results of the algorithm for the example circuit

4.2 Reconvergent fanouts

When a logic signal splits into multiple branches (fanout) and later reconverges in two or more inputs of a gate, the subnet thus formed is referred as *reconvergent fanout*. Reconvergent fanouts are an important issue in areas dealing with signal probability such as testing [29] or reliability, since they break the assumption that signal paths are independent. Moreover, it has been documented that in current VLSI designs, about half the nodes in the circuit cause reconvergence [33].

Computing signal probabilities using equations such as (4.5) or (4.6) is referred to as *0-Algorithm*. In simple circuits without reconvergent fanout nodes (RFON), the computation of the signal probabilities by the 0-Algorithm is exact. In Figure 4.4, the inaccuracies caused by assuming independence are shown. Input 2 has a fanout of 2, and is divided into A_1 and A_2. Both paths (B_2 and C) reconverge at gate G_2. Figure 4.4a shows the results of propagating the input probabilities P_i using the 0-Algorithm. Using the same circuit, we conduct a Monte Carlo experiment: thousands of input vectors are sampled from the input probabilities P_i and the value at the output of each gate is recorded. Then, the 0-to-1 signal ratio is obtained for the output of every gate. These probabilities are the ones provided in Figure 4.4b. It can be seen that the probabilities that do not involve a reconvergent fanout (G_1, G_3 and

(a) Probabilities computed by the 0-Algorithm



(b) Observed probabilities

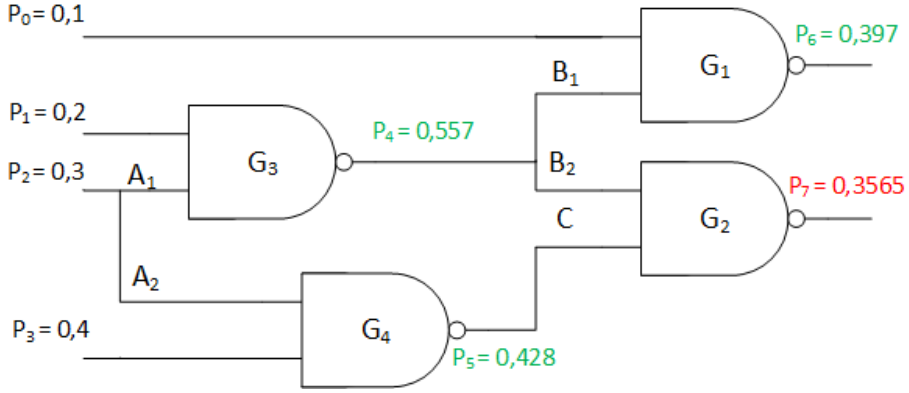


Fig. 4.4 Error caused by reconvergent fanouts

G_4) are correctly computed by the 0-Algorithm, while the probability at the output of the reconvergence (G_2) is not.

Within the literature, there are a few widely-adopted, polynomial-time algorithms that try to reduce the lack of precision induced by the 0-Algorithm [24], [35], [1]. While each approach tackles the problem differently, they are based under the observation that the signal probability $p(j)$ of a node j can be expressed by:

$$p(j) = p(j|f = 0) * p(f = 0) + p(j|f = 1) * p(f = 1) \tag{4.10}$$

where f is a RFON of the fanin cone in j. If f were the only RFON in the circuit, $p(j)$ could be exactly calculated by using equations such as (4.6) or (4.7) and applying three times the 0-Algorithm: first to evaluate $p(f)$, then, forcing the logical value of f to 0 and 1 to calculate

the conditional probabilities. The approaches in [24] and [1] use these expressions in the circuit's primary inputs, while the algorithm in [35] estimates the effect that each RFON in the circuit has in every signal. As detailed in Section 6.1.1, the use of these algorithms improves the precision of the computation in comparison with the 0-Algorithm, but it still is insufficient if we want them to be applied in a signal-probability based reliability algorithm.

We propose to use the heuristics in [1] (the ones, described in Appendix A, that in our implementation had the best results) as a first step and then exhaustively simulate a critical subset of reconvergent subnets. Our research shows that, within the benchmark circuits, more than 80% of the subnets have 30 or less gates. Even though the error introduced by larger subnets is not negligible, correcting a large amount of relatively small subnets provided good results in terms of accuracy with little overhead in execution time (simulating the few million combinations of all subnets of size ≤ 30 can be done in milliseconds. Comprehensive numbers are given in Section 6.1.2). In order to perform these simulations, a static search of all RFON is performed using Roberts' algorithm to detect reconvergent fanouts [36], detailed in Appendix B.

Our approach to compute signal probabilities, which corresponds to Line 1 in Algorithm 1 is summarized in Figure 4.5.

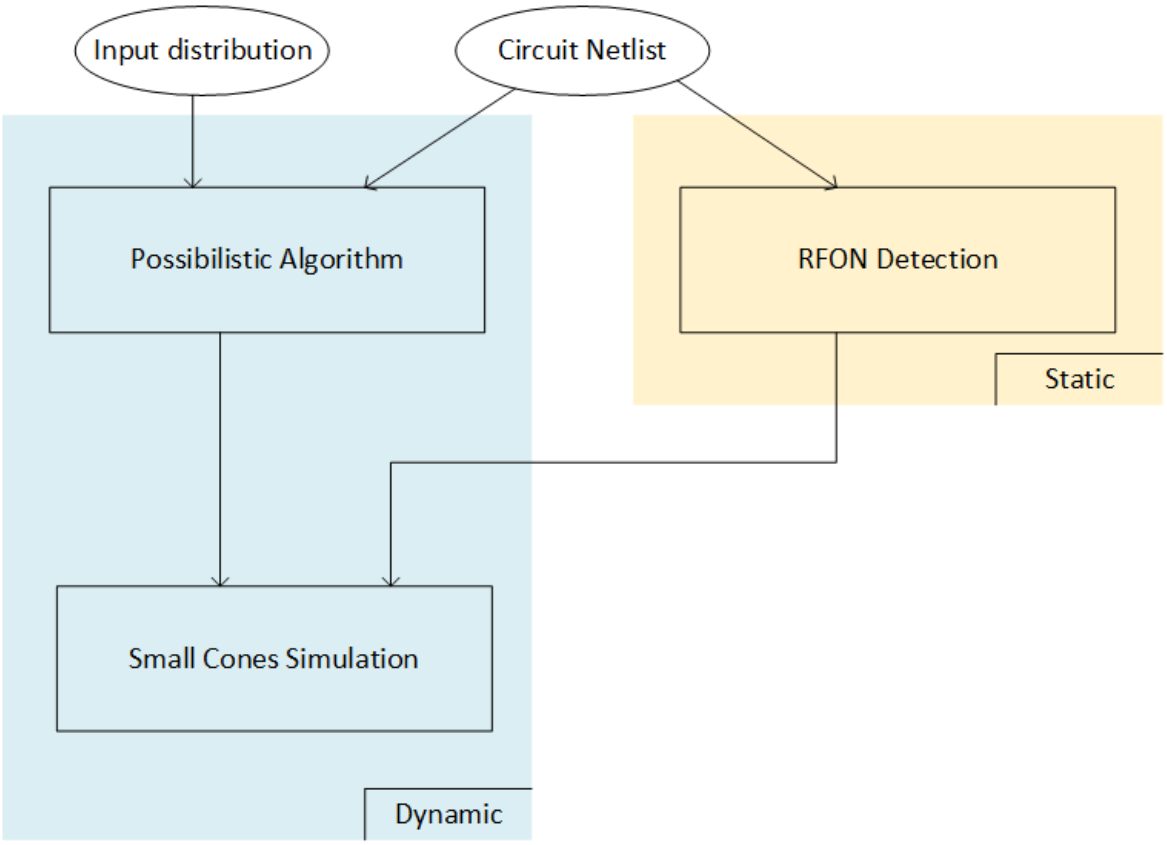


Fig. 4.5 Methodology to compute signal probabilities

Chapter 5

Validation methodology

This chapter explains the fault injection algorithm used to validate our reliability estimation methodology and describes the benchmarks used

In order to test our algorithm, we compare the output of the algorithm with the results of fault injection experiments. The experiments consist of three variables: the input probability distribution, the circuit upon which the injection is performed and number of iterations. The input distribution consists of a random number between 0 and 1 for each primary input of the circuit. The circuits can be divided in two sets: one set of small circuits compiled from similar works in the literature (described in Table 5.1) and the ISCAS85 benchmarks, ten widely-accepted combinational circuits provided at the 1985 International Symposium on Circuits And Systems[1] (described in Table 5.2).

Table 5.1 Small benchmark circuits

Circuit	Total gates	Primary inputs	Primary outputs
c17 (Fig. 5.1)	6	5	2
Full adder	5	3	2
2x4 decoder	6	2	4
Logic chain (Fig. 5.2)	7	5	1
4-bit ripple carry adder	20	9	5

The fault injection consists in changing the bit at the output of a gate. To perform an experiment, a particular input vector from an input distribution is generated and the circuit is simulated. Then, faults are injected iteratively to all gates and outputs are sampled for changes with respect to the original simulation. If there are no differences among the original

[1] Even though the benchmarks are 30 years old, these circuits are widely used for testing and reliability purposes, since they contain parts of the processor, such as adders or ECCs, that are still valid nowadays.

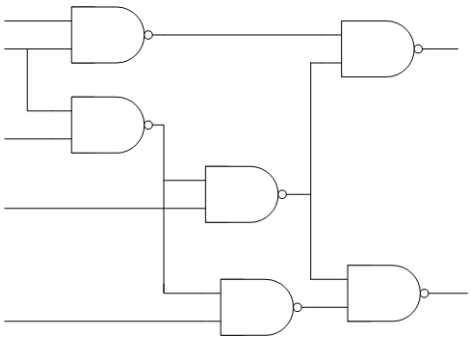


Fig. 5.1 Small circuit c17

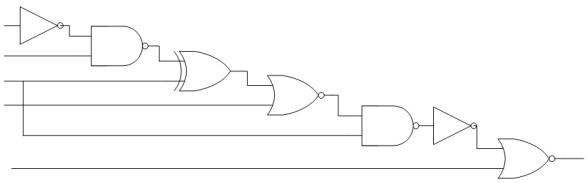


Fig. 5.2 Small logic chain

Table 5.2 ISCAS85 benchmark circuits

Circuit name	Circuit function	Total gates	Primary inputs	Primary outputs
c432	Priority decoder	160	36	7
c499	32-bit SEC circuit	202	41	32
c880	8-bit ALU	383	60	26
c1355	32-bit SEC circuit	546	41	32
c1908	16-bit SEC/DED circuit	880	33	25
c2670	12-bit ALU	1193	233	140
c3540	8-bit ALU	1669	50	22
c5315	9-bit ALU	2307	178	123
c6288	16-bit multiplier	2406	32	32
c7552	32-bit adder and comparator	3512	207	108

outputs and the fault-injected ones, the fault was masked. If one or more outputs have been changed, the fault was not masked and a counter, representing vulnerability, for the gate is incremented. This process is repeated for a number of iterations, inside which new input vectors are sampled from the input distribution. When a sufficiently large number of input vectors have been simulated, the vulnerability counter at each of the gates is divided by the number of iterations, thus computing the ratio of experiments in which a fault in the gate was not masked, i.e., its vulnerability. These actions are repeated for 100 different input probability distributions. The procedure is formalized in Algorithm 2.

The variable $GATES$ in Line 3 and in Line 17 corresponds to the number of gates in the circuit, while the variable $EXPERIMENTS$ in Line 6 and in Line 18 corresponds to the number of iterations of the experiment. The number of samples in a Montecarlo experiment is defined by equation (5.1) [37].

$$n = \frac{z_{\alpha/2}^2 * S_n^2}{\varepsilon^2} \tag{5.1}$$

where:

- n is the number of iterations needed.
- $z_{\alpha/2}$ is the critical value of the normal distribution for $\alpha/2$, the z value such that the area of the right-hand tail is $\alpha/2$.
- S_n is the estimated standard deviation of the output.
- ε is the desired margin of error.

For our experiments, we choose a interval of confidence of 95%, so $z_{\alpha/2} = 1.96$. Since the output of the experiment are probability numbers representing vulnerability, we choose a margin of error of 1%. Even though we do not know the standard deviation of our population, it can be estimated by a large number (such as 10^7 [42]) of beforehand preliminary simulations. These simulations yield an estimated standard deviation of 0.32. Therefore, according to Equation 5.1, the number of Monte Carlo trials is 3934. We round this number to 4000 iterations.

Since technology information is not yet available to us, we assume all transition probabilities $T_{gate}^{out}[0 \rightarrow 1]$ and $T_{gate}^{out}[1 \rightarrow 0]$ described in Algorithm 1 of Section 4.1 to be 1. This will overestimate the vulnerability of the circuits because not all strikes cause a bit-flip [2], but it

[2]For example, a strike in a 2-input NAND gate collected by a n-type drain while the both inputs are 0 does not cause the output to change

Algorithm 2 Fault injection experiments

```
1: for i ← 1 to 100 do
2:     inputDistribution ← Generate random numbers between 0 and 1
3:     for j ← 1 to GATES do
4:         Vulnerability[j] ← 0
5:     end for
6:     for i ← 1 to EXPERIMENTS do
7:         inputValues ← Generate random values from inputDistribution
8:         originalOutputs ← evaluateCircuit(inputValues)
9:         for j ← 1 to GATES do
10:            Change output of gate j
11:            faultInjectionOutputs ← evaluateCircuitWithStrike(j, inputValues)
12:            if faultInjectionOutputs ≠ originalOutputs then
13:                Vulnerability[j] ← Vulnerability[j] + 1
14:            end if
15:        end for
16:    end for
17:    for j ← 1 to GATES do
18:        Vulnerability[j] ← Vulnerability[j]/EXPERIMENTS
19:    end for
20: end for
```

does not affect the validation since it is only a scaling factor that behaves identically in the fault injector and in the model.

Chapter 6

Results

This chapter contains the results of the validation experiments between the model and fault injection approaches, as well as a comparison among the literature algorithms and our method to compute signal probability.

6.1 Signal probability computation

In the presented algorithm, signal probabilities are used to evaluate logical masking because it is straightforward to compute the logical masking of a gate knowing the ratio of 0 and 1 signals in its input vectors. Therefore, to properly measure the effect of logical masking, we must properly calculate signal probabilities. The complexity of computing signal probabilities is a #P-complete problem, so heuristics have to be used.

6.1.1 Literature algorithms

Using the 0-Algorithm, i.e., estimating probabilities assuming signal independence, has to be discarded due to the large number of RFON in current circuits. We reviewed the literature for solutions for this problem and found three widely-adopted algorithms:

- Weighted Averaging Algorithm (WAA): Corrects the effects of RFON at the primary inputs by forcing zero and one to the inputs, and averaging the resulting probabilities by an heuristic weight.

- Dynamic WAA: Computes the signal probability of each node taking into account the effect of all RFON by iterating through them and forcing zeros and ones.

- Possibilistic Algorithm: Computes several possible signal probabilities for each node and then heuristically chooses one depending on the distance of each number to the 0-Algorithm probability.

Since, to the best of our knowledge, numbers comparing the methods are not provided, we implemented all of them and compared their effectiveness. To do so, random input vectors were sampled from an input distribution and the value at each node was noted. After 10^7 simulation runs, the reference ratio between 0 and 1 (the probability that there is a 0 at that node) at each node is obtained. Then, the different algorithms, whose outputs are signal probabilities per node, are executed. Finally, the difference between the reference probability and the algorithm probability is measured.

We used circuits from the ISCAS85 benchmarks to conducts these experiments. 100 different input distributions were tested to extract conclusions. For brevity purposes, there are only 4 input distributions shown per circuit. In the following tables, the mean of the absolute difference between the simulated probabilities and the computed probabilities for each algorithm is reported. Adjacent to each table is a graph plotting the results.

	0 Alg.	WAA	Possibilistic Alg.	Dynamic WAA
Input distribution 1	0,030	0,030	0,010	0,031
Input distribution 2	0,051	0,049	0,021	0,051
Input distribution 3	0,048	0,048	0,031	0,048
Input distribution 4	0,059	0,056	0,036	0,058

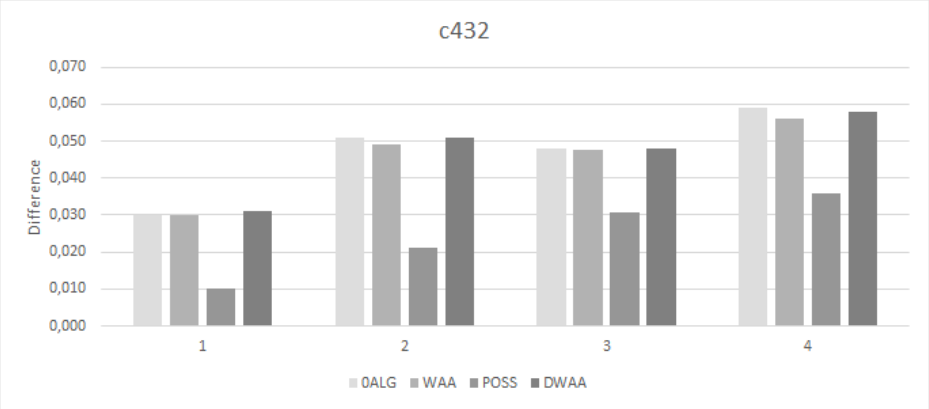


Fig. 6.1 Signal probability difference comparison for c432

	0 Alg.	WAA	Possibilistic Alg.	Dynamic WAA
Input distribution 1	0,002	0,002	0,001	0,002
Input distribution 2	0,002	0,002	0,001	0,002
Input distribution 3	0,002	0,002	0,001	0,002
Input distribution 4	0,005	0,005	0,005	0,002

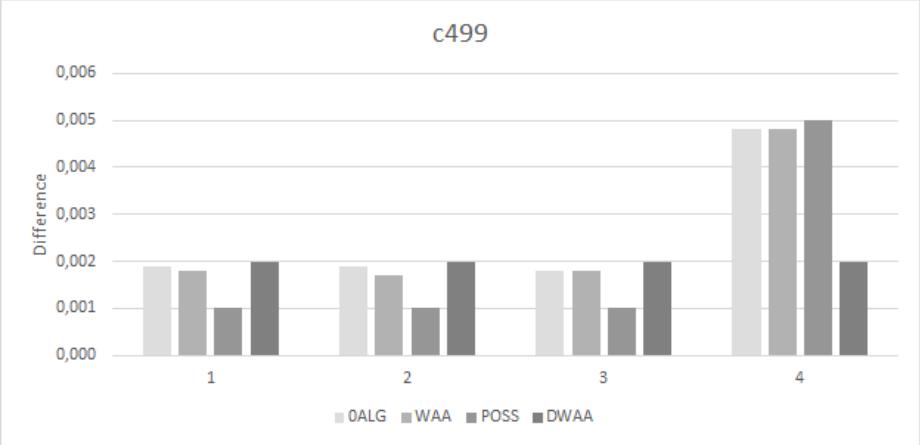


Fig. 6.2 Signal probability difference comparison for c499

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,010	0,008	0,008	0,009
Input distribution 2	0,008	0,008	0,008	0,007
Input distribution 3	0,014	0,013	0,010	0,012
Input distribution 4	0,012	0,011	0,010	0,013

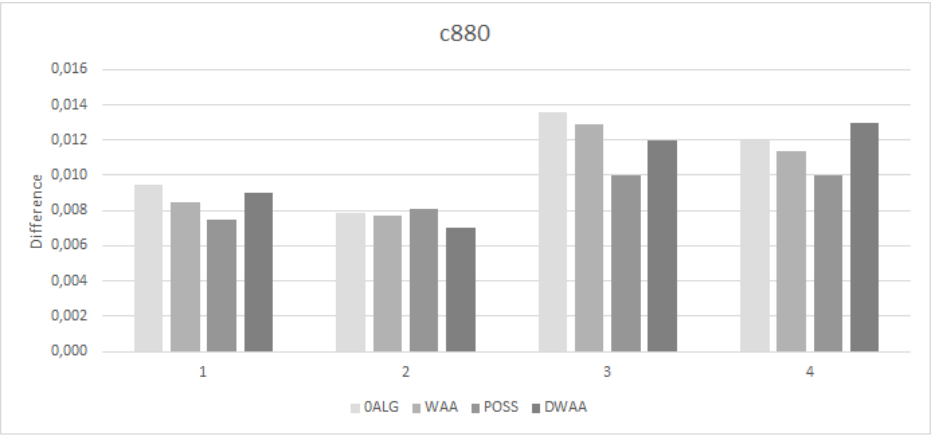


Fig. 6.3 Signal probability difference comparison for c880

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,065	0,047	0,039	0,064
Input distribution 2	0,057	0,044	0,037	0,055
Input distribution 3	0,063	0,046	0,036	0,060
Input distribution 4	0,059	0,044	0,037	0,050

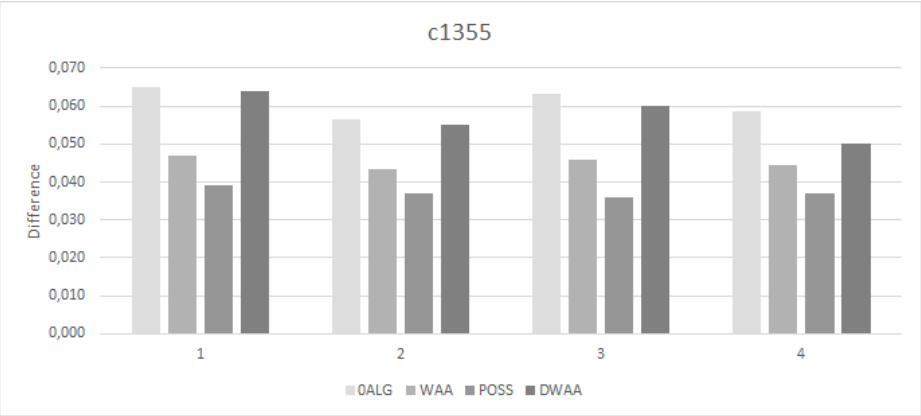


Fig. 6.4 Signal probability difference comparison for c1355

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,017	0,048	0,012	0,017
Input distribution 2	0,020	0,054	0,015	0,020
Input distribution 3	0,023	0,048	0,017	0,022
Input distribution 4	0,022	0,046	0,019	0,021

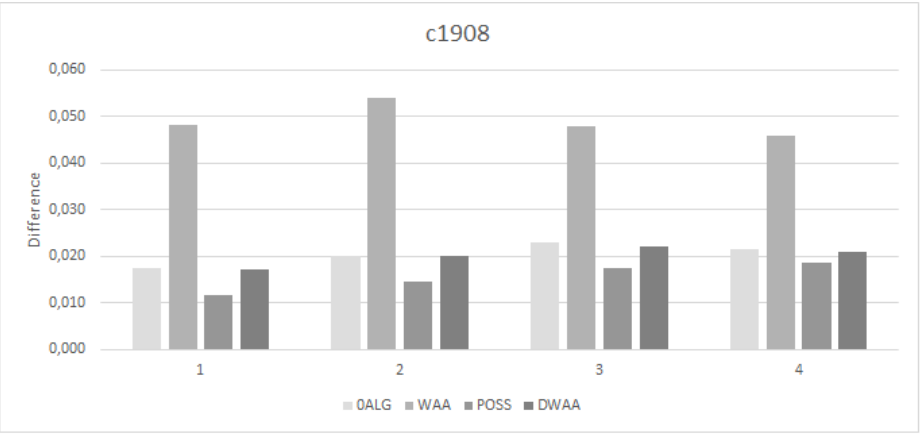


Fig. 6.5 Signal probability difference comparison for c1908

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,043	0,073	0,030	0,041
Input distribution 2	0,040	0,076	0,032	0,038
Input distribution 3	0,043	0,084	0,034	0,042
Input distribution 4	0,039	0,066	0,031	0,038

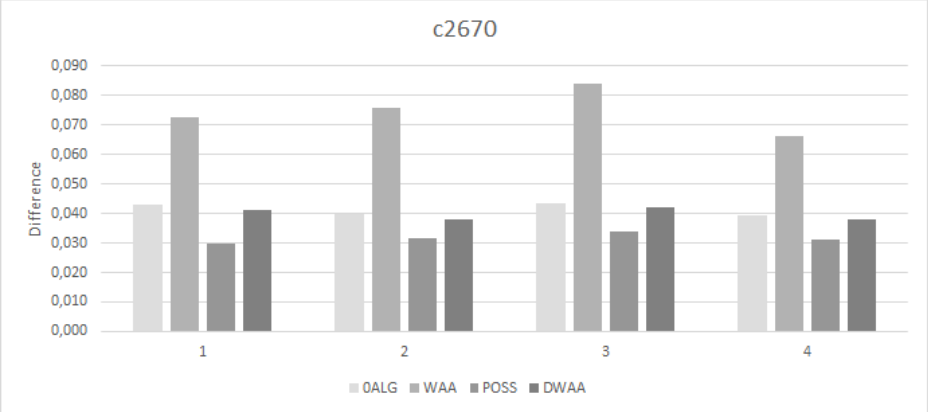


Fig. 6.6 Signal probability difference comparison for c2670

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,045	0,085	0,040	0,043
Input distribution 2	0,042	0,077	0,026	0,041
Input distribution 3	0,046	0,094	0,028	0,045
Input distribution 4	0,043	0,080	0,037	0,043

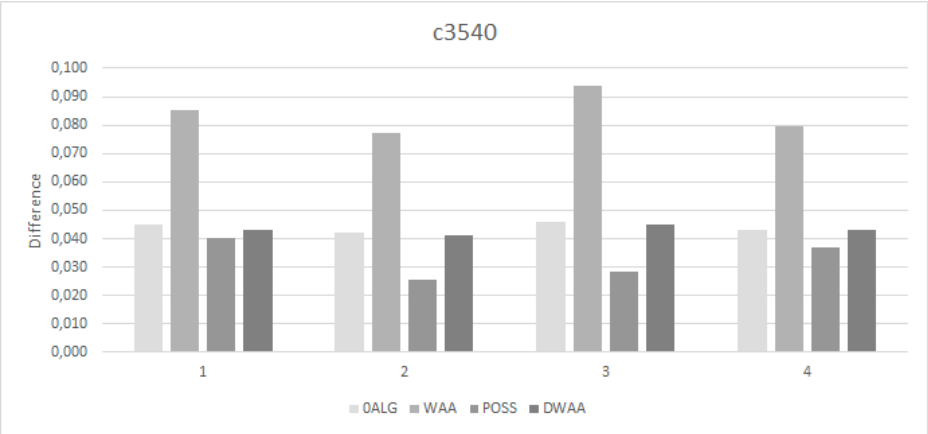


Fig. 6.7 Signal probability difference comparison for c3540

	0 Algorithm	WAA	Possibilistic Algorithm	Dynamic WAA
Input distribution 1	0,059	0,076	0,044	0,056
Input distribution 2	0,045	0,071	0,032	0,043
Input distribution 3	0,040	0,074	0,040	0,059
Input distribution 4	0,068	0,084	0,038	0,065

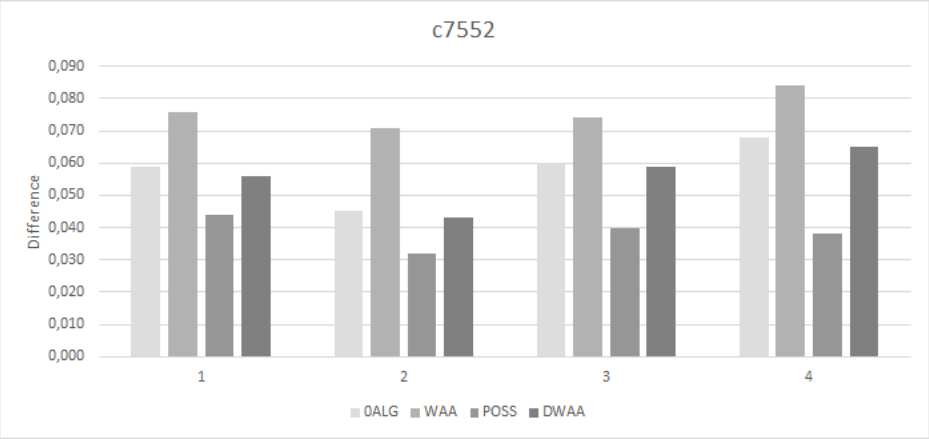


Fig. 6.8 Signal probability difference comparison for c7552

Results show that, in the majority of cases, the Possibilistic algorithm yields significantly better results than the 0-Algorithm (average of 50%) and the other two algorithms found in the literature (average of 30%). Of the 1000 cases tested, in only 6 the Possibilistic algorithm performed worse than any of the other three. A couple of examples are depicted: for circuit c499 (Fig. 6.2, fourth input distribution) and for circuit c880 (Fig.6.3, second input distribution).

The use of the Possibilistic algorithm reduces greatly the signal probability computation error but, in average, it still differs from the correct probabilities by 0.021. Although it is not a large number, we consider that it is not an acceptable error because of the important role that signal probabilities have in our reliability estimation algorithm.

6.1.2 Proposed approach

For a given fanout cone, we define its inputs as the leads reaching the cone and coming from nodes not contained in the cone. We propose to generate the 2^n inputs, simulate the circuit 2^n times and average the 0-to-1 ratios of the nodes by the likelihood of each input vector. The simulation time for a particular cone could be high since it depends on the number of inputs reaching the cone. We have checked the ISCAS85 benchmarks and found that:

- Of 10000 fanout cones within the circuits, more than 80% have 30 nodes or less.

- The average number of inputs in fanout cones of at most size 30 is 21.
- Each circuit has an average of 950 islands.
- Simulating 2^{21} inputs is a matter of a few milisecons.

Figure 6.9 shows a trade-off between signal probability and exectution time depending on the maximum cone size studied. Within large cones, gates with big fanin counts (5 or more) are more present, and the number of inputs reaching the cone increases dramatically with the cone size. Moreover, despite the extra computation effort, the differences between the simulated probabilities and the computed probabilities are not reduced significantly.

We conclude that simulating the small cones is feasible in terms of execution time (negligible amount of seconds) and it provides significant improvement in the computation of signal probabilities. Therefore, to compute signal probabilities, we first execute the Possibilistic algorithm and then we exhaustively simulate all fanout cones that have 30 nodes or less. This results in an average error of 0.001, 21 times less than the error of just executing the Possibilistic algorithm. We consider that an error of 0.1% in the computation of signal probabilities is good enough to build our vulnerability estimation upon.

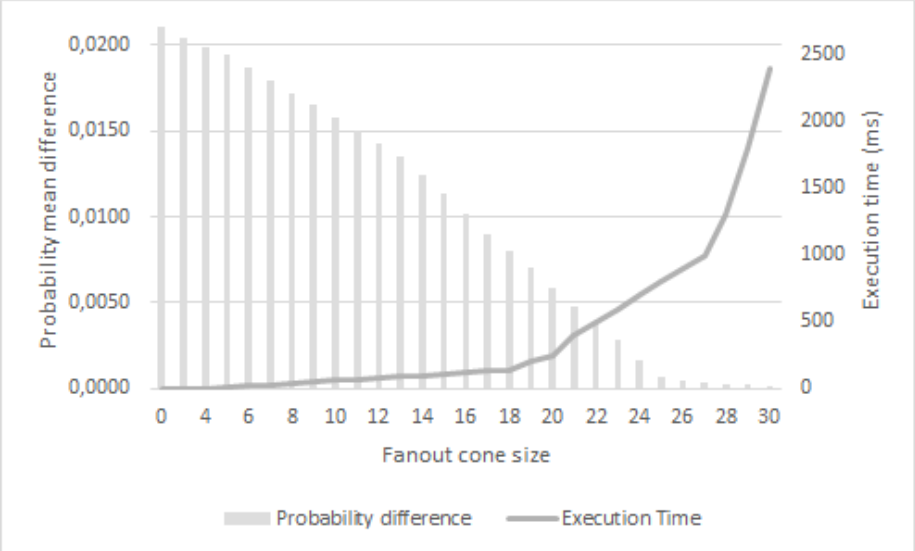


Fig. 6.9 Trade-off between exhaustively simulating cones up to a particular size and execution time

6.2 Vulnerability computation

In this section, we compare our model (Algorithm 1 of Section 4.1) using the method proposed in Section 6.1.2 for computing signal probabilities with the Monte Carlo fault injection approach described in Algorithm 2 of Chapter 5. 100 different input distributions were tested to extract conclusions. For brevity purposes, there are only 4 input distributions shown per circuit. In the following tables, the mean of the vulnerability of all nodes is shown for the fault injection experiments and model executions (*Vulnerability*). Additionally, the table shows, as a figure of absolute error, the mean of absolute value of the difference between the fault injection experiment and the model execution of each node (*Gate mean difference*) and the relative error between the model and the fault injection experiment. Adjacent to each table is a graph plotting the results: the left axis shows the vulnerability of the fault injection and the model and the relative error is measured in the right axis.

$$Vulnerability = \frac{\sum_{i=1}^{nodes} vulnerability(i)}{nodes} \tag{6.1}$$

$$GateMeanDifference = \frac{\sum_{i=1}^{nodes} \left| vulnerability_{Model}(i) - vulnerability_{FaultInjection}(i) \right|}{nodes} \tag{6.2}$$

In Section 6.2.1, we present the results for the set of small circuits compiled from similar works in the literature. In Section 6.2.2, we present the results for the ISCAS85 benchmarks.

6.2.1 Small benchmarks from the literature

c17		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,864	0,003	3,2
	Model	0,892		
Input distribution 2	Fault Injection	0,825	0,027	3,3
	Model	0,853		
Input distribution 3	Fault Injection	0,811	0,003	0,2
	Model	0,812		
Input distribution 4	Fault Injection	0,742	0,033	4,3
	Model	0,774		

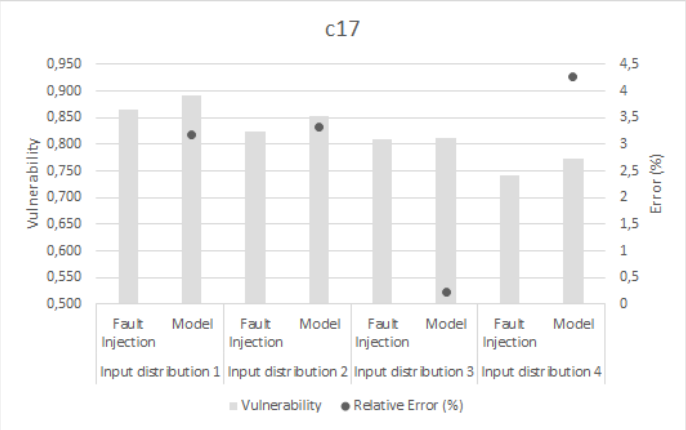


Fig. 6.10 Fault injection and Model comparison for c17

Logic chain		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,154	0,001	0,3
	Model	0,154		
Input distribution 2	Fault Injection	0,222	0,004	1,8
	Model	0,226		
Input distribution 3	Fault Injection	0,165	0,001	0,9
	Model	0,167		
Input distribution 4	Fault Injection	0,395	0,015	2,5
	Model	0,385		

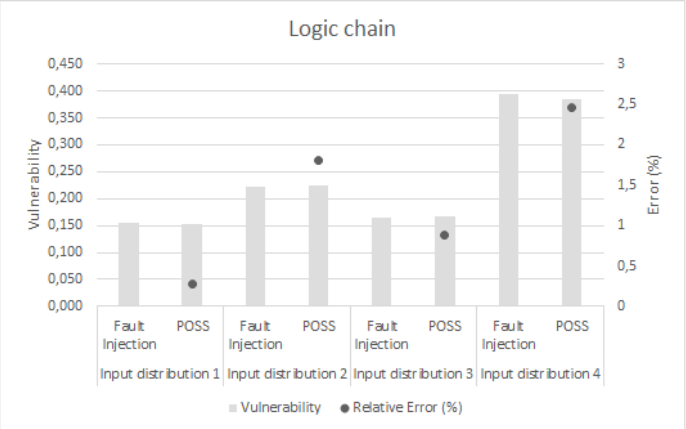


Fig. 6.11 Fault injection and Model comparison for a small logic chain

Decoder		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	1	0,048	1,8
	Model	0,982		
Input distribution 2	Fault Injection	1	0,082	1,8
	Model	0,982		
Input distribution 3	Fault Injection	1	0,024	1,4
	Model	0,986		
Input distribution 4	Fault Injection	1	0,049	1,9
	Model	0,981		

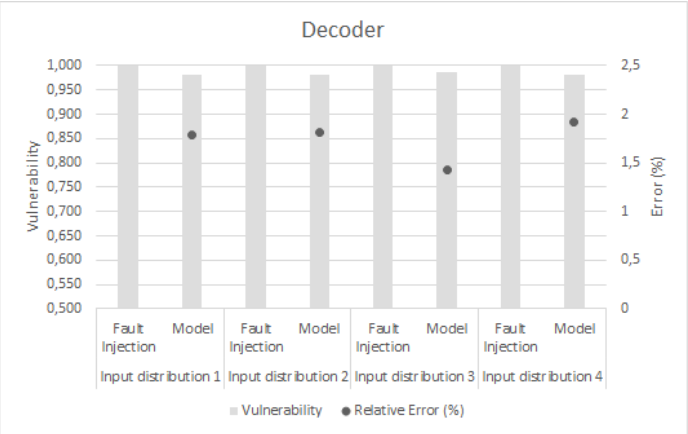


Fig. 6.12 Fault injection and Model comparison for a 2x4 decoder

Full adder		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,907	0,005	1,8
	Model	0,905		
Input distribution 2	Fault Injection	0,973	0,014	1,3
	Model	0,960		
Input distribution 3	Fault Injection	0,811	0,047	1,4
	Model	0,799		
Input distribution 4	Fault Injection	0,899	0,026	1,6
	Model	0,875		

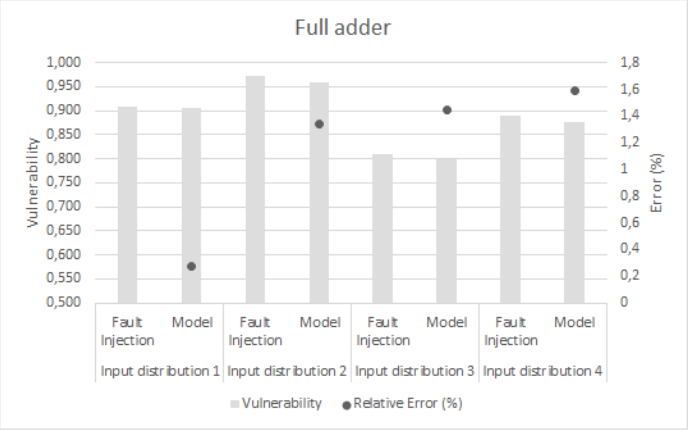


Fig. 6.13 Fault injection and Model comparison for a full adder

4-bit adder		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,931	0,090	5,1
	Model	0,883		
Input distribution 2	Fault Injection	0,930	0,079	5,2
	Model	0,881		
Input distribution 3	Fault Injection	0,861	0,068	1,7
	Model	0,847		
Input distribution 4	Fault Injection	0,860	0,079	2,3
	Model	0,840		

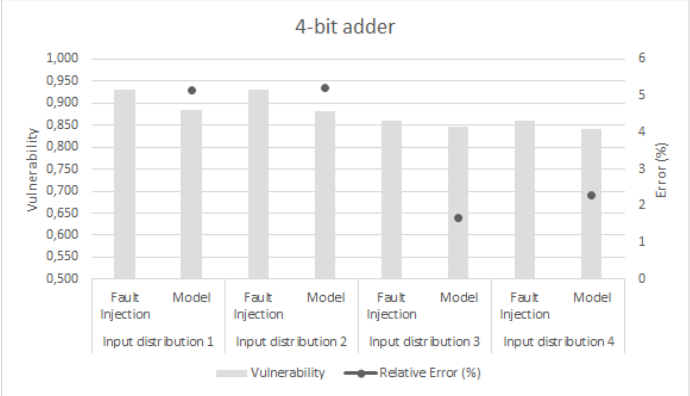


Fig. 6.14 Fault injection and Model comparison for a 4-bit adder

Results for the circuits presented show an average error of 2% between the fault injection experiments and the model execution. The effect that the input values have in logical masking can be clearly observed in Figure 6.11, where different inputs can almost triple vulnerability. In the rest of the circuits the effect is not as clearly seen due to high vulnerability values. That is because circuits as the Decoder (Fig. 6.12) that have almost all its gates connected to the output have high vulnerability since the majority of the strikes are not logically masked, whereas circuits with several levels such as the Logic chain (Fig. 6.11) are more likely to mask strikes near the primary inputs.

Fault injection experiments in small experiments like these had an average execution time of 11 seconds, while the average execution time of the model was 183 miliseconds.

6.2.2 ISCAS85 benchmarks

c432		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,241	0,032	4,7
	Model	0,230		
Input distribution 2	Fault Injection	0,253	0,031	1,6
	Model	0,257		
Input distribution 3	Fault Injection	0,334	0,003	0,8
	Model	0,331		
Input distribution 4	Fault Injection	0,255	0,033	5,1
	Model	0,268		

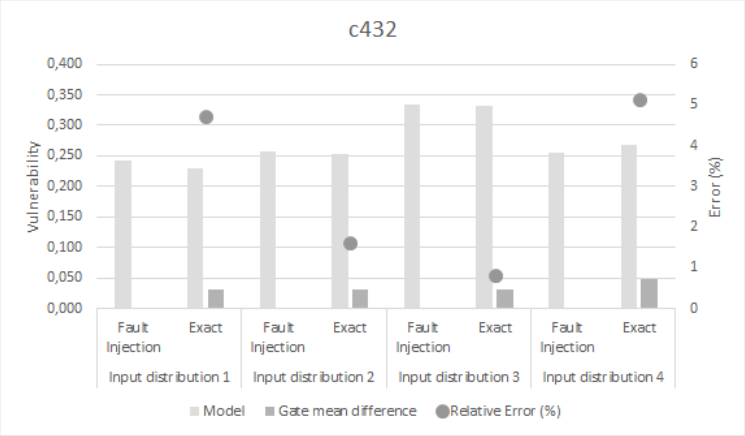


Fig. 6.15 Fault injection and Model comparison for c432

c499		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,4675	0,023	0,004
	Model	0,4675		
Input distribution 2	Fault Injection	0,4670	0,022	0,004
	Model	0,4670		
Input distribution 3	Fault Injection	0,4660	0,020	0,009
	Model	0,4660		
Input distribution 4	Fault Injection	0,4674	0,019	0,011
	Model	0,4674		

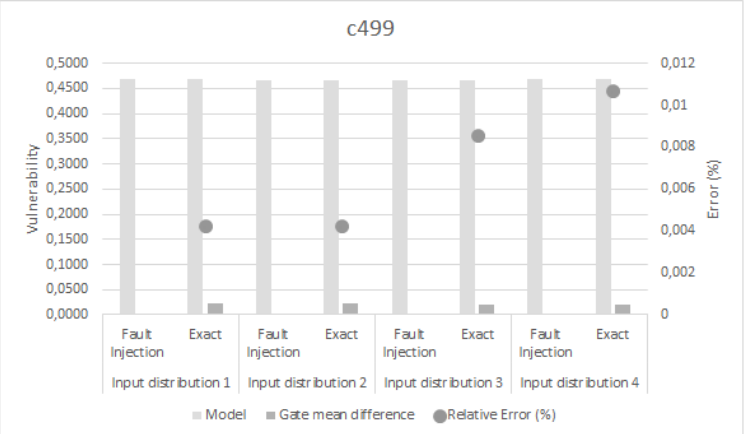


Fig. 6.16 Fault injection and Model comparison for c499

c880		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,556	0,099	16,2
	Model	0,466		
Input distribution 2	Fault Injection	0,531	0,085	14,3
	Model	0,455		
Input distribution 3	Fault Injection	0,472	0,088	16,0
	Model	0,396		
Input distribution 4	Fault Injection	0,591	0,103	15,5
	Model	0,499		

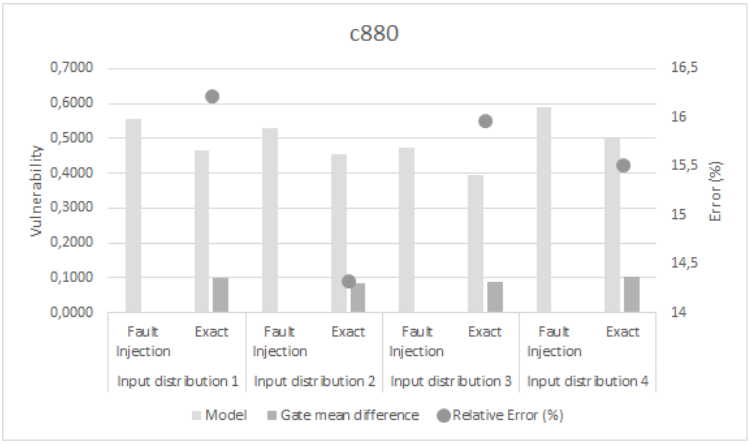


Fig. 6.17 Fault injection and Model comparison for c880

c1355		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,384	0,049	8,8
	Model	0,351		
Input distribution 2	Fault Injection	0,381	0,044	9,0
	Model	0,350		
Input distribution 3	Fault Injection	0,380	0,046	8,2
	Model	0,353		
Input distribution 4	Fault Injection	0,3857	0,043	7,3
	Model	0,3566		

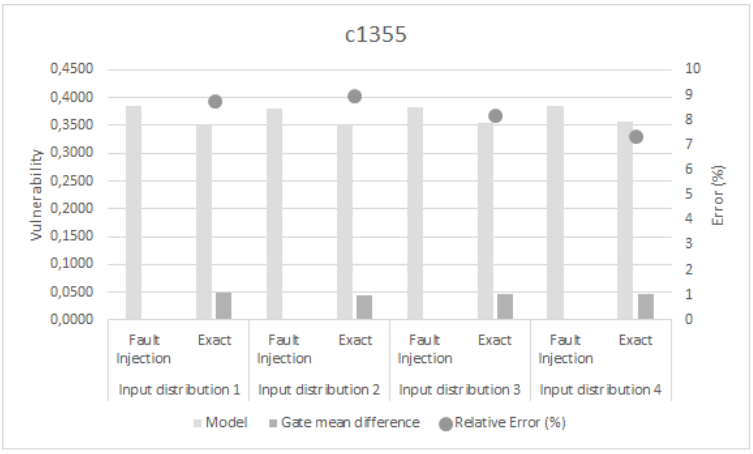


Fig. 6.18 Fault injection and Model comparison for c1355

c1908		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,398	0,14	32,5
	Model	0,267		
Input distribution 2	Fault Injection	0,462	0,17	26,5
	Model	0,292		
Input distribution 3	Fault Injection	0,450	0,17	29,9
	Model	0,279		
Input distribution 4	Fault Injection	0,387	0,14	37,2
	Model	0,250		

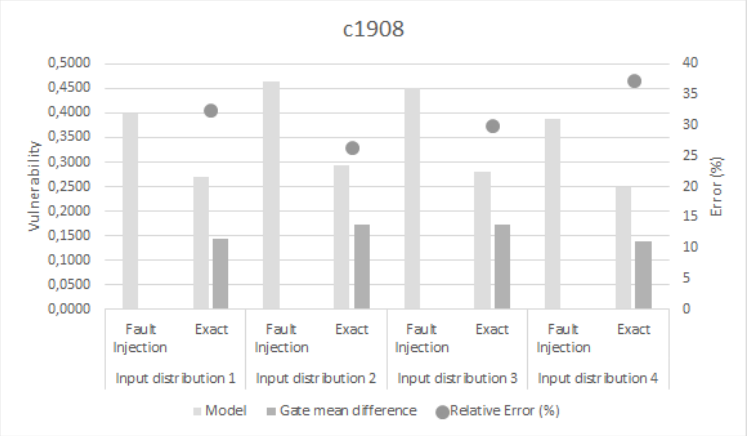


Fig. 6.19 Fault injection and Model comparison for c1908

c2670		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,361	0,11	27,0
	Model	0,263		
Input distribution 2	Fault Injection	0,341	0,10	31,1
	Model	0,249		
Input distribution 3	Fault Injection	0,305	0,07	31,4
	Model	0,237		
Input distribution 4	Fault Injection	0,410	0,14	22,0
	Model	0,281		

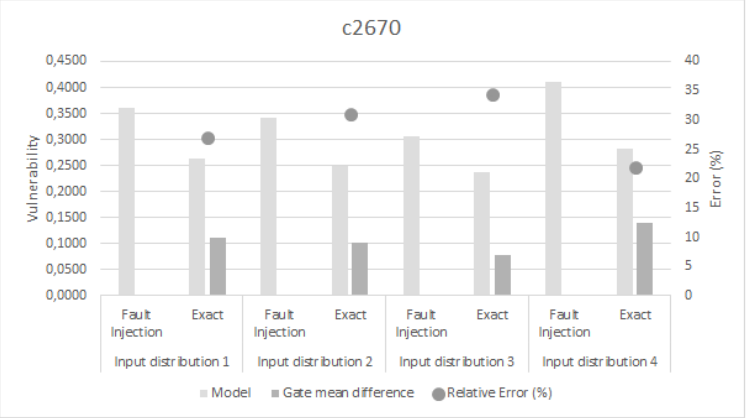


Fig. 6.20 Fault injection and Model comparison for c2670

c3540		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,275	0,13	40,8
	Model	0,162		
Input distribution 2	Fault Injection	0,280	0,14	47,6
	Model	0,144		
Input distribution 3	Fault Injection	0,298	0,14	41,2
	Model	0,162		
Input distribution 4	Fault Injection	0,275	0,12	41,5
	Model	0,161		

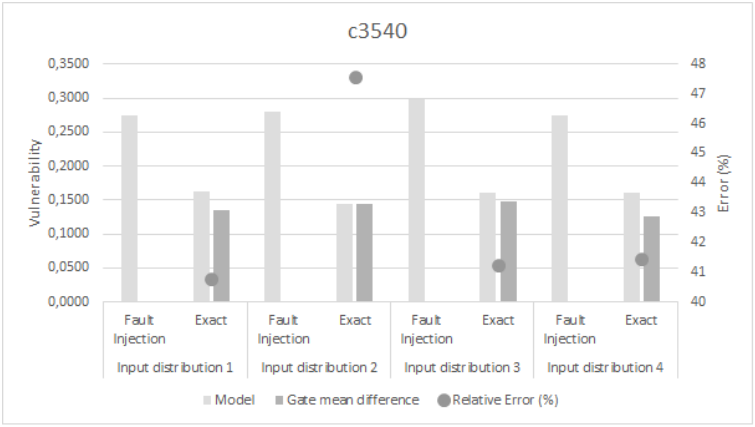


Fig. 6.21 Fault injection and Model comparison for c3540

c7552		Vulnerability	Gate mean diff.	Relative Error (%)
Input distribution 1	Fault Injection	0,385	0,176	42,6
	Model	0,221		
Input distribution 2	Fault Injection	0,374	0,177	44,7
	Model	0,207		
Input distribution 3	Fault Injection	0,375	0,174	42,9
	Model	0,214		
Input distribution 4	Fault Injection	0,385	0,172	41,7
	Model	0,224		

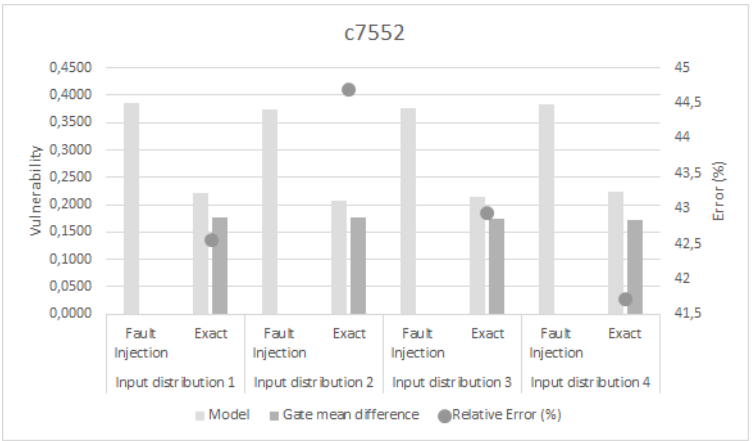


Fig. 6.22 Fault injection and Model comparison for c7552

There are two sets of results: the first half of the benchmarks, circuits c432 through c1355, and the second half of the benchmarks.

Within the first half (Figures 6.15, 6.16, 6.17, 6.18), the average relative error is 4% while the maximum is 16%. In circuits c432 and c499, the absolute error is 1%; in circuit c880 is 8% and, in circuit c1355 is 4%. These results are good, considering that the execution time with respect to the fault injection approach is reduced from hours to seconds, as shown in Table 6.1.

Within the second half of the benchmarks (Figures 6.19, 6.20, 6.21, 6.22) the results are not that optimistic: while execution time is reduced from days to seconds, the average relative error is 31%. Moreover, the mean differences per gate between the fault injection experiment and the model are as high as 0.2.

It is clear that those errors are not related to the size of the circuit. In fact, there is less error in c499 than in c432, there is less error in c1355 than in c880 and there is less error in c7552 than in c5315. We believe that the assumption of independence when combining the vulnerabilities of succeeding gates (Line 15 in Algorithm 1) is what causes that error.

Table 6.1 Execution time comparison between fault injection and our model in ISCAS85 benchmarks

Circuit	Fault injection execution time	Model execution time
c432	1 hour	3 seconds
c499	1.5 hours	5 seconds
c880	4.7 hours	8 seconds
c1355	10 hours	16 seconds
c1908	22 hours	26 seconds
c2670	40 hours	30 seconds
c3540	3 hours	41 seconds
c5315	7.5 days	59 seconds
c7552	1.5 weeks	2 minutes

Chapter 7

Conclusions and Future Work

This chapter summarizes the contributions and achievements of this research and outlines the future research directions from this work.

7.1 Conclusions

With continuously shrinking transistor dimensions but constant cosmic neutron flux at the ground level, the soft error vulnerability of semiconductor devices is increasing. Hardening mechanisms for combinational logic are expensive in terms of performance loss, power consumption and area and, therefore, they must be used sparsely and only in critical parts of the circuit. Processor designers need tools that can accurately compute what components demand protection from soft errors, but these tools are required to be fast in order to be integrated inside the architectural design flow. Techniques that accomplish these objectives are not easy to develop because of the difficulty of the problem and the large size of today's integrated circuits.

We have proposed a reliability estimation model based on signal probabilities. Because it is an analytical model, it does not need to use time consuming fault injection or simulate multiple input vectors because the effect of the signal values is inherently taken into account, unlike most existing models. The algorithm presented computes the effect of logical masking and can easily be extended to also consider electrical masking and latch-window masking without significant loss in execution time. Because of its low use in input parameters and straightforward implementation, it could be combined with other architectural tools.

The results on small circuits are comparable to the ones found in the literature in terms of accuracy (average error of 2%) and speed (average execution time of 50 ms). The results

on the subset of ISCAS85 benchmarks that have a low RFON count can also be positively matched against the literature, with an average error of 4%, a maximum error of 8% and an average execution time of 10 seconds. We are producing reliability estimations as good as other works with greater speedup and a much simpler methodology. For ISCAS85 benchmarks with a large number of RFON, our algorithm, while fast, has an average error of 31%. We believe the cause is combining the masking effects and vulnerabilities of the successors of a node assuming the successors will have different paths towards the outputs.

The errors caused by the independence assumption have been solved in this thesis for the problem of computing signal probability. We have combined a widely-accepted algorithm in the literature called Possibilistic algorithm with exhaustive circuit simulation on the most common fanout cones. A large fraction of the fanout cones of a circuit are small, which allows for a fast, in-depth simulation in order to obtain the real probabilities for every node in the cone. Our experiments show that with a few million cone simulations, corresponding to less than 3 seconds, the error of the signal probability computation is reduced to 0.1%. This translates to an estimation more than 20 times better than the one the Possibilistic algorithm provides.

7.2 Future work

7.2.1 Error reduction

Our algorithm is fast and has good accuracy in circuits with a small number of dependencies. If we want to be able to introduce the model within an architectural tool, we must reduce the large error produced by that circuits with lots of dependencies. We will begin by describing the different scenarios in which there is a dependence and perform several simulations to analyze them. With the conclusions we extract from the analysis, we will derive useful heuristics to use in our model.

7.2.2 Electrical masking and latch-window masking

The algorithm shown in this thesis computes the probability that a strike in a connection has an impact in any output by traversing the circuit in such way that information about logical masking can be incrementally gained. Using the same structure, electrical and window-latching masking effects can be included. The gates should have additional information relevant to these effects, such as delay or size analogously to the masking equations the algorithm currently uses at each gate to compute the logical masking effect. Regarding

latch-window masking, a strike in a connection will only cause a soft error if the duration of the pulse is sufficiently long and it arrives at a time during the clock cycle that allows it to reach a latch at the output of the circuit during its latching window. With information about delay in each gate, the latching-window masking probability can be computed for every gate. Regarding electrical masking, a strike in a connection will only cause a soft error if the amplitude of the pulse is enough to be latched, taking into account the degradation suffered until it reaches an output. With information about degradation of the amplitude, the electrical masking probability can be computed for every gate.

For the addition of the other masking effects, physical characteristics of the pulses, such as amplitude and duration, are needed to perform the calculations. There is, therefore, a need to design a technology library that contains such information. Since we already defined the use of some technology-level data in the transition probabilities and we have still not included them in our model, the characterization needed for the electrical and latch-window masking is along the lines of our research.

7.2.3 Multiple particle strikes

Complexity can be added to the functions such as the ones described in equations 4.5, 4.6 or 4.7, to include the event of more than one particle striking. For example, following the 2-input NAND gate example of Section 4.1:

1. If both inputs are 0, the output will be affected only if both inputs are changed by strikes.
2. If one input is 1 and the other is 0:
   (a) If the input set to 1 suffers a transition from 1 to 0, the output is not affected.
   (b) If the input set to 0 suffers a transition from 0 to 1, the output changes only if the other input does not change.
3. If both inputs are 1, a transition from 1 to 0 at one or both inputs changes the output.

References

[1] Al-Kharji, M., Al-Arian, S., et al. (1997). A new heuristic algorithm for estimating signal and detection probabilities. In *VLSI, 1997. Proceedings. Seventh Great Lakes Symposium on*, pages 26–31. IEEE.

[2] Amerasekera, A., Ramaswamy, S., Chang, M.-C., and Duvvury, C. (1996). Modeling mos snapback and parasitic bipolar action for circuit-level esd and high current simulations. In *Reliability Physics Symposium, 1996. 34th Annual Proceedings., IEEE International*, pages 318–326. IEEE.

[3] Baumann, R. (2005). Soft errors in advanced computer systems. *Design & Test of Computers, IEEE*, 22(3):258–266.

[4] Baumann, R. C. (2001). Soft errors in advanced semiconductor devices: Part i: The three radiation sources. *IEEE Transactions on Device and Materials Reliability*, 1(1):17–22.

[5] Belov, V. (2007). Radiation hardened latch. US Patent 7,212,056.

[6] Black, J. R. (1969). Electromigration—a brief survey and some recent results. *Electron Devices, IEEE Transactions on*, 16(4):338–347.

[7] Blat, C., Nicollian, E., and Poindexter, E. (1991). Mechanism of negative-bias-temperature instability. *Journal of Applied Physics*, 69(3):1712–1720.

[8] Boudreaux, D., Williams, F., and Nozik, A. (1980). Hot carrier injection at semiconductor-electrolyte junctions. *Journal of Applied Physics*, 51(4):2158–2163.

[9] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691.

[10] Chang, A. C.-C., Huang, R. H.-M., and Wen, C. H.-P. (2013). Casser: a closed-form analysis framework for statistical soft error rate. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(10):1837–1848.

[11] Dodd, P. E. and Massengill, L. W. (2003). Basic mechanisms and modeling of single-event upset in digital microelectronics. *Nuclear Science, IEEE Transactions on*, 50(3):583–602.

[12] Edmonds, L. D. (1998). Electric currents through ion tracks in silicon devices. *Nuclear Science, IEEE Transactions on*, 45(6):3153–3164.

[13] Edmonds, L. D. (2001). A time-dependent charge-collection efficiency for diffusion. *Nuclear Science, IEEE Transactions on*, 48(5):1609–1622.

[14] Franco, D. T., Vasconcelos, M. C., Naviner, L., and Naviner, J.-F. (2008). Signal probability for reliability evaluation of logic circuits. *Microelectronics Reliability*, 48(8):1586–1591.

[15] Gao, J., Qi, Y., and Fortes, J. A. (2005). Bifurcations and fundamental error bounds for fault-tolerant computations. *Nanotechnology, IEEE Transactions on*, 4(4):395–402.

[16] Garey, M. R. and Johnson, D. S. (1979). Computers and intractability: a guide to np-completeness.

[17] Han, J., Chen, H., Liang, J., Zhu, P., Yang, Z., and Lombardi, F. (2014). A stochastic computational approach for accurate and efficient reliability evaluation. *Computers, IEEE Transactions on*, 63(6):1336–1350.

[18] Holcomb, D., Li, W., and Seshia, S. A. (2009). Design as you see fit: System-level soft error analysis of sequential circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 785–790. European Design and Automation Association.

[19] Hsieh, C. M., Murley, P. C., and O'brien, R. R. (1983). Collection of charge from alpha-particle tracks in silicon devices. *Electron Devices, IEEE Transactions on*, 30(6):686–693.

[20] Hsueh, M.-C., Tsai, T. K., and Iyer, R. K. (1997). Fault injection techniques and tools. *Computer*, 30(4):75–82.

[21] Jensen, F. V. (1996). *An introduction to Bayesian networks*, volume 210. UCL press London.

[22] Kehl, N. and Rosenstiel, W. (2011). An efficient ser estimation method for combinational circuits. *Reliability, IEEE Transactions on*, 60(4):742–747.

[23] Kim, J. S., Nicopoulos, C., Vijaykrishnan, N., Xie, Y., and Lattanzi, E. (2004). A probabilistic model for soft-error rate estimation in combinational logic. In *Proc. of the Int'l Workshop on Probabilistic Analysis Techniques for Real-time and Embedded Systems*.

[24] Krishnamurthy, B. and Tollis, I. G. (1989). Improved techniques for estimating signal probabilities. *Computers, IEEE Transactions on*, 38(7):1041–1045.

[25] Kurd, N., Chowdhury, M., Burton, E., Thomas, T. P., Mozak, C., Boswell, B., Mosalikanti, P., Neidengard, M., Deval, A., Khanna, A., et al. (2015). Haswell: A family of ia 22 nm processors. *Solid-State Circuits, IEEE Journal of*, 50(1):49–58.

[26] Kwon, S., Park, J. K., and Kim, J. T. (2014). An approximated soft error analysis technique for gate-level designs. *IEICE Electronics Express*, 11(10):20140224–20140224.

[27] Lyons, R. E. and Vanderkulk, W. (1962). The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209.

[28] Murley, P. C. and Srinivasan, G. (1996). Soft-error monte carlo modeling program, semm. *IBM Journal of Research and Development*, 40(1):109–118.

[29] Nigh, P. and Maly, W. (1990). Test generation for current testing (cmos ics). *Design & Test of Computers, IEEE*, 7(1):26–38.

[30] Ong, T. (2001). Tsmc 6t-sram soft error rate (ser) summary. *TSMC*.

[31] Pagliarini, S. N., Dhia, A. B., Naviner, L. d. B., and Naviner, J.-F. (2013). Snap: A novel hybrid method for circuit reliability assessment under multiple faults. *Microelectronics Reliability*, 53(9):1230–1234.

[32] Peterson, W. W. and Weldon, E. J. (1972). *Error-correcting codes*. MIT press.

[33] Ratiu, I. M., Sangiovanni-Vincentelli, A. L., and Pederson, D. O. (1982). Victor: A fast vlsi testability analysis program. In *ITC*, pages 397–403.

[34] Rejimon, T., Lingasubramanian, K., and Bhanja, S. (2009). Probabilistic error modeling for nano-domain logic circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):55–65.

[35] Ricc, B. (1989). Estimate of signal probability in combinational logic networks.

[36] Roberts, M. and Lala, P. (1987). Algorithm to detect reconvergent fanouts in logic circuits. *Computers and Digital Techniques, IEE Proceedings E*, 134(2):105–111.

[37] Robey, R. R. and Barcikowski, R. S. (1992). Type i error and the number of iterations in monte carlo studies of robustness. *British Journal of Mathematical and Statistical Psychology*, 45(2):283–288.

[38] Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., and Alvisi, L. (2002). Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 389–398. IEEE.

[39] Singh, J., Mohanty, S. P., and Pradhan, D. (2012). *Robust SRAM Designs and Analysis*. Springer Science & Business Media.

[40] Synopsys (2015). Hspice, accurate circuit simulation. [Online; accessed 26-June-2015].

[41] Wang, F. and Xie, Y. (2006). An accurate and efficient model of electrical masking effect for soft errors in combinational logic. In *Proc. Second Workshop System Effects of Logic Soft Error (SELSE2)*.

[42] Winston, W. L. (2000). *Simulation modeling using@ RISK*. Duxbury.

[43] Zhang, B., Wang, W.-S., and Orshansky, M. (2006). Faser: Fast analysis of soft error susceptibility for cell-based designs. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 755–760. IEEE Computer Society.

[44] Zhang, M. and Shanbhag, N. R. (2006). Soft-error-rate-analysis (sera) methodology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(10):2140–2155.

[45] Ziegler, J. F. (1996). Terrestrial cosmic rays. *IBM journal of research and development*, 40(1):19–39.

Appendix A

Possibilistic algorithm to estimate signal probability

Step 1: Execute 0-Algorithm to estimate the signal probability S at the output node of each gate.

Step 2: For each primary input I_i, perform the following:

1. Execute 0-Algorithm setting I_i to 0, obtaining a signal probability $0Alg(I_i = 0)$ at the output of each gate.

2. Execute 0-Algorithm setting I_i to 1, obtaining a signal probability $0Alg(I_i = 1)$ at the output of each gate.

3. Compute the estimate p_i at the output node of each gate by averaging $0Alg(I_i = 0)$ and $0Alg(I_i = 1)$ by the signal input probability p_i of input I_i:

$$p_i = p_{I_i} * 0Alg(I_i = 0) * (1 - p_{I_i}) * 0Alg(I_i = 1)$$

Every gate now has a probability tuple (P-tuple), containing estimates from each primary input.

Step 3: Calculate the level of every gate in the network and sort the gates in non-decreasing order according to their levels.

Step 4: Proceeding from the inputs to the outputs of each gate, perform the following steps from the gates with the lowest level:

1. Compute the expected tuple (E-tuple) of each output gate using the P-tuples of its inputs and independent equations such as (4.5) or (4.6).

2. For every component p_i in the P-tuple, compare it with S. If they are equal, mark it with 0. If they are not, then compare with its expected value e_i from the E-tuple and mark it with 1 if they are equal and with 2 otherwise. By doing this, a mark tuple (M-tuple) is created.

3. Count the number of 1s and 2s in the M-tuple, denoted by $ctr1$ and $ctr2$, respectively.

4. Count the no-dependent (ND) value by using independent equations and the estimated signal probabilities P of the previous level.

5. Apply the following inference rules to compute the probability P at the output of each gate:

if ($ctr1 = 0$ and $ctr2 = 0$) **then** $P = S$
if ($ctr1 \neq 0$ and $ctr2 = 0$) **then** $P = ND$
if ($ctr1 = 0$ and $ctr2 \neq 0$) **then**{
  if ($ctr2 = 1$) **then** $P = p_i$
  else{
    $effect_2 = \sum_{m_i=2} (p_i - e_i)$ †
    $P = ND + effect_2 + sign(effect_2) * \frac{effect2}{ctr2}$ ‡
  }
}
if ($ctr1 \neq 0$ and $ctr2 \neq 0$) **then**{
  $effect_2 = \sum_{m_i=2} (p_i - e_i)$
  $P = ND + effect_2 + sign(effect_2) * (S - ND) * \frac{effect2}{ctr2}$
}

† $m_i = 2$ corresponds to all components that are marked with a 2 in the M-tuple.
‡ *sign* corresponds to a function that returns the sign of its argument.

Appendix B

Algorithm to detect reconvergent fanouts in logic circuits

Every node N of the circuit has associated a fanout list (FOL). The FOL contains a list of entries for every fanout node that, directly or indirectly, feeds N. Each entry consists of two components: a unique identifier for the fanout node and a path count which holds the number of pats between the fanout node and N. In addition, each node in the circuit has a reconvergent fanout list (RFOL). The RFOL of a node contains an entry for every fanout that reconverges at that node.

Algortithm to detect reconvergent fanouts

Step 1: Read the circuit description file

Step 2: For each node N in the circuit:

$reached(N) \leftarrow$ number of input lines feeding N

Step 3:

1. $NL \leftarrow$ list of primary inputs —Next List
2. $CL \leftarrow NL$ —Current List
3. $NL \leftarrow \varnothing$

Step 4: For each node $N1$ in CL:

For each node $N2$ fed by $N1$:

$reached(N2) \leftarrow$ reached($N2$)-1

if ($reached(N2) = 0$) **then** $NL \leftarrow NL + N2$

Step 5: For every node N in CL:

buildFols(N) —detailed below

reduceRFOL(N) —detailed below

Step 6: For each node N in CL which is a fanout node:
Generate a unique identifier for the fanout
Place identifier (with a path count of 1) in the FOL of node N
Step 7: **if** ($NL = \varnothing$) **then** go to Step 3.2
Step 8: Output RFOLs of the nodes

buildFOLs(node N)

Step 1: $RFOL(N) \leftarrow \varnothing$
Step 2: $FOL(N) \leftarrow FOL(Input_1(N))$
Step 3: For all inputs $Input_x(N)$ of N except $Input_1(N)$:
$RFOL(N) \leftarrow RFOL(N) \cup (FOL(N) \cap FOL(Input_x(N)))$
$FOL(N) \leftarrow FOL(Z) \cup FOL(Input_x(N))$

reduceRFOL(node N)

Step 1: $newRFOL(N) \leftarrow \varnothing$
Step 2: **while** $RFOL(N) \neq \varnothing$ **do**
$Y \leftarrow$ fanout with largest identifier in $RFOL(N)$
$newRFOL(N) \leftarrow newRFOL(N) \cup Y$
$k \leftarrow$ path count for fanout Y
$RFOL(N) \leftarrow RFOL(N) - k * FOL(Y)$