Chapter 1:

PROBABILISTIC ERROR PROPAGATION IN A LOGIC CIRCUIT USING THE BOOLEAN DIFFERENCE CALCULUS

Nasir Mohyuddin
Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089, USA
mohyuddi@usc.edu

Ehsan Pakbaznia
Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089, USA
pakbazni@usc.edu

Massoud Pedram
Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089, USA
pedram@usc.edu

Table of Contents

Abstract

A gate level probabilistic error propagation model is presented which takes as input the Boolean function of the gate, input signal probabilities, the error probability at the gate inputs, and the gate error probability and generates the error probability at the output of the gate. The presented model uses the Boolean difference calculus and can be efficiently applied to the problem of calculating the error probability at the primary outputs of a multi-level Boolean circuit with a time complexity which is linear in the number of gates in the circuit. This is done by starting from the primary inputs and moving toward the primary outputs by using a post-order—reverse Depth First Search (DFS)—traversal. Experimental results demonstrate the accuracy and efficiency of the proposed approach compared to the other known methods for error calculation in VLSI circuits.

Keywords

Signal probability, Error probability, Co-factor, Partial Boolean Difference, Re-convergent fanout, Spatial correlations.

1.1 Introduction

As CMOS hits nano-scale regime, device failure mechanisms such as cross talk, manufacturing variability, and soft error become significant design concerns. Being probabilistic by nature, these failure sources have pushed the CMOS technology toward stochastic CMOS [1]. For example, capacitive and inductive coupling between parallel adjacent wires in nano-scale CMOS Integrated Circuits (ICs) are the potential sources of crosstalk between the wires. Crosstalk can indeed cause flipping error on the *victim* signal [2]. In addition to the probabilistic CMOS, promising nanotechnology devices such as quantum dots are used in technologies such as Quantum Cellular Automata (QCA). Most of these emerging technologies are inherently probabilistic. This has made reliability analysis an essential piece of circuit design. . Reliability analysis will be even more significant in designing reliable circuits using unreliable components [3][4].

Circuit reliability will thus be an important tradeoff factor which has to be taken care of similar to traditional design tradeoff factors such as performance, area, and power. To include the reliability into the design tradeoff equations, there must exist a good measure for the circuit reliability, and there must exist fast and robust tools that, similar to timing analyzer and power estimator tools, are capable of estimating circuit reliability at different design levels. In [5] authors have proposed a Probabilistic Transfer Matrix (PTM) method to calculate the output signal error probability for a circuit while [6] presents a method based on the Probabilistic Decision Diagrams (PDDs) to perform this task.

In this chapter we first introduce a probabilistic gate level error propagation model based on the concept of Boolean difference to propagate errors from inputs to output of a general gate. We then apply this model to account for the error propagation in a given circuit and finally estimate the error probability at the circuit outputs. Note that in the proposed model a gate's Boolean function is used to determine the error propagation in the gate. An error at an output of a gate is due to its input(s) and/or the gate itself being erroneous. The internal gate error in this work is modeled as an output flipping event. This means that, when a faulty gate makes an error, it flips (changes a "1" to a "0" and a "0" to a "1") its output value that it would have generated given the inputs, Von Neumann error model. In the rest of this chapter, we call our circuit error estimation technique the Boolean Difference-based Error Calculator, or BDEC for short, and we assume that a defective logic gate produces the wrong output value for every input combination. This is a more pessimistic defect model than the stuck-at-fault model.

Authors in [5] use a PTM matrix for each gate to represent the error propagation from the input(s) to the output(s) of a gate. They also define some operations such as matrix multiplication and tensor product to use the gate PTMs to generate and propagate error probability at different nodes in a circuit level-by-level. Despite of its accuracy in calculating signal error probability, PTM technique suffers from the extremely large number of computational-intensive tasks namely regular and tensor matrix products. This makes the PTM technique extremely memory intensive and very slow. In particular, for larger circuits, size of the PTM matrices grows too fast for the deeper nodes in circuit making PTM an inefficient or even infeasible technique of error rate calculation for a general circuit. References [8] and [9] developed a methodology based on probabilistic model checking (PMC) to evaluate the circuit reliability. The issue of excessive memory requirement of PMC when the circuit size is large was successfully addressed in [10]. However, the time complexity still remains a problem. In fact, the authors of [10] show that the run time for their space-efficient approach is even worse than that of the original approach.

Boolean difference calculus was introduced and used by [11] and [12] to analyze single faults. It was then extended by [13] and [14] to handle multiple fault situations, however, they only consider stuck-at-faults and they do not consider the case when the logic gates themselves can be erroneous and hence a gate-induced output error may nullify the effect of errors at the gate's input(s). In [15] authors use Bayesian networks to calculate the output error probabilities without considering the input signal probabilities.

The author in [6] uses probabilistic decision diagrams (PDD) to calculate the error probabilities at the outputs using probabilistic gates. While PDDs are much more efficient than PTM for average case, the worst-case complexity of both PTM and PDD-based error calculators is exponential in the number of inputs in the circuit.

In contrast, we will show in section 1.5 that BDEC calculates the circuit error probability much faster than PTM while achieving as accurate results as PTM's. We will show that BDEC requires a single pass over the circuit nodes using a post-order (reverse DFS) traversal to calculate the errors probabilities at the output of each gate as

we move from the primary inputs to the primary outputs; hence, complexity is O (N) where N is the number of the gates in the circuit, and O (.) is the big O notation.

1.2 Error Propagation Using Boolean Difference Calculus

Some key concepts and notation that will be used in the remainder of this chapter are discussed next.

1.2.1 Partial Boolean difference

The partial Boolean difference of function $f(x_1, x_2, \ldots, x_n)$ with respect to one variable or a subset of its variables [14] is defined as:

$$\frac{\partial f}{\partial x_i} = f_{x_i} \oplus f_{\bar{x}_i}$$
$$\frac{\partial f}{\partial x_{i_1} x_{i_2} \ldots x_{i_k}} = \frac{\partial f}{\partial \left(x_{i_1} x_{i_2} \ldots x_{i_k} \right)} = f_{x_{i_1} x_{i_2} \ldots x_{i_k}} \oplus f_{\bar{x}_{i_1} \bar{x}_{i_2} \ldots \bar{x}_{i_k}} \tag{1.1}$$

where $\oplus$ represents XOR operator and f_{x_i} is the co-factor of f with respect to x_i, i.e.,

$$f_{x_i} = f(x_1, \ldots, x_{i-1}, x_i = 1, x_{i+1}, \ldots, x_n)$$
$$f_{\bar{x}_i} = f(x_1, \ldots, x_{i-1}, x_i = 0, x_{i+1}, \ldots, x_n) \tag{1.2}$$

Higher order co-factors of f can be defined similarly. The partial Boolean difference of f with respect to x_i expresses the condition (with respect to other variables) under which f is sensitive to a change in the input variable x_i. More precisely, if the logic values of $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$ are such that $\partial f/\partial x_i = 1$, then a change in the input value x_i, will change the output value of f.. However, when $\partial f/\partial x_i = 0$, changing the logic value of x_i will not affect the output value of f.

It is worth mentioning that the *order-k partial Boolean difference* defined in Equation 1.1 is different from the k^{th} *Boolean difference* of function f as used in [13], which is denoted by $\partial^k f / \partial x_{i_1} \ldots \partial x_{i_k}$. For example, the 2nd Boolean difference of function f with respect to x_i and x_j is defined as:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j} \right) = f_{x_i x_j} \oplus f_{\bar{x}_i x_j} \oplus f_{x_i \bar{x}_j} \oplus f_{\bar{x}_i \bar{x}_j} \tag{1.3}$$

Therefore, $\partial^2 f/\partial x_i \partial x_j \neq \partial f/\partial (x_i x_j)$.

1.2.2 Total Boolean difference

Similar to the partial Boolean difference that shows the conditions under which a Boolean function is sensitive to change of any of its input variables, we can define *total Boolean difference* showing the condition under which the output of the Boolean function f is sensitive to the *simultaneous* changes in all the variables of a subset of

input variables. For example, the total Boolean difference of function *f* with respect to $x_i x_j$ is defined as:

$$\frac{\Delta f}{\Delta(x_i x_j)} = \frac{\partial f}{\partial(x_i x_j)}\left(x_i x_j + \overline{x}_i \overline{x}_j\right) + \frac{\partial f}{\partial(\overline{x}_i x_j)}\left(\overline{x}_i x_j + x_i \overline{x}_j\right) \tag{1.4}$$

where $\Delta f/\Delta(x_i x_j)$ describes the conditions under which the output of *f* is sensitive to a simultaneous change in x_i and x_j. That is, the value of *f* changes as a result of the simultaneous change. Some examples for simultaneous changes in x_i and x_j are transitioning from $x_i=x_j=1$ to $x_i=x_j=0$ and vice versa, or from $x_i=1$, $x_j=0$ to $x_i=0$, $x_j=1$ and vice versa. However, transitions in the form of $x_i=x_j=1$ to $x_i=1$, $x_j=0$ or $x_i=1$, $x_j=0$ to $x_i=0$, $x_j=0$ are not simultaneous changes. Note that $\partial f/\partial(x_i x_j)$ describes the conditions when a transition from $x_i=x_j=1$ to $x_i=x_j=0$ and vice versa changes the value of function *f*.

It can be shown that the total Boolean difference in Equation 1.4 can be written in the form of:

$$\frac{\Delta f}{\Delta\left(x_i x_j\right)} = \frac{\partial f}{\partial x_i} \oplus \frac{\partial f}{\partial x_j} \oplus \frac{\partial^2 f}{\partial x_i \partial x_j} \tag{1.5}$$

The total Boolean difference with respect to three variables is:

$$\begin{aligned}\frac{\Delta f}{\Delta(x_1 x_2 x_3)} = {} & \frac{\partial f}{\partial(x_1 x_2 x_3)}\left(x_1 x_2 x_3 + \overline{x}_1 \overline{x}_2 \overline{x}_3\right) \\ & + \frac{\partial f}{\partial(x_1 x_2 \overline{x}_3)}\left(x_1 x_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 x_3\right) \\ & + \frac{\partial f}{\partial(x_1 \overline{x}_2 x_3)}\left(x_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3\right) \\ & + \frac{\partial f}{\partial(\overline{x}_1 x_2 x_3)}\left(\overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3\right)\end{aligned} \tag{1.6}$$

It is straightforward to verify that:

$$\begin{aligned}\frac{\Delta f}{\Delta(x_1 x_2 x_3)} = {} & \frac{\partial f}{\partial x_1} \oplus \frac{\partial f}{\partial x_2} \oplus \frac{\partial f}{\partial x_3} \oplus \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ & \oplus \frac{\partial^2 f}{\partial x_2 \partial x_3} \oplus \frac{\partial^2 f}{\partial x_1 \partial x_3} \oplus \frac{\partial^3 f}{\partial x_1 \partial x_2 \partial x_3}\end{aligned} \tag{1.7}$$

In general total Boolean difference of a function *f* with respect to an n-variable subset of its inputs can be written as:

$$\frac{\Delta f}{\Delta(x_{i_1} x_{i_2} \ldots x_{i_n})} = \sum_{j=0}^{2^{n-1}-1} \left.\frac{\partial f}{\partial \vec{x}}\right|_{m_j} \left(m_j + m_{2^n-j-1}\right) \tag{1.8}$$

where m_j's are defined as follows:

$$\begin{aligned} m_0 &= \overline{x}_{i_1}\overline{x}_{i_2}\ldots\overline{x}_{i_{n-1}}\overline{x}_{i_n} \\ m_1 &= \overline{x}_{i_1}\overline{x}_{i_2}\ldots\overline{x}_{i_{n-1}}x_{i_n} \\ &\vdots \\ m_{2^n-1} &= x_{i_1}x_{i_2}\ldots x_{i_{n-1}}x_{i_n}, \end{aligned} \tag{1.9}$$

and we have:

$$\left.\frac{\partial f}{\partial \vec{x}}\right|_{m_j} = \frac{\partial f}{\partial\left(x_1^* x_2^* \ldots x_{n-1}^* x_n^*\right)} \text{ where } m_j = x_1^* x_2^* \ldots x_{n-1}^* x_n^* \tag{1.10}$$

1.2.3 Signal and error probabilities

Signal probability is defined as the probability for a signal value to be "1". That is:

$$p_i = \Pr\{x_i = 1\} \tag{1.11}$$

Gate error probability is shown by ε_g and is defined as the probability that a gate generates an erroneous output, independent of its applied inputs. Such a gate is sometimes called $(1-\varepsilon_g)$-reliable gate. Signal error probability is defined as the probability of error on a signal line. If the signal line is the output of a gate, the error can be either due to error at the gate input(s) or the gate error itself. We denote the error probability on signal line x_i by ε_i.

We are interested in determining the circuit output error rates, given the circuit input error rates under the assumption that each gate in the circuit can fail independently with a probability of ε_g. In other words, we account for the general case of multiple simultaneous gate failures.

1.3 Proposed Error Propagation Model

In this section we propose our gate error model in the Boolean difference calculus notation. The gate error model is then used to calculate the error probability and reliability at outputs of a circuit.

1.3.1 Gate Error Model

Figure 1.1 shows a general logic gate realizing Boolean function *f*, with gate error probability of ε_g. The signal probabilities at the inputs, i.e., probabilities for input signals being 1, are $p_1, p_2, \ldots, p_n$ while the input error probabilities are $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$. The output error probability is ε_z.

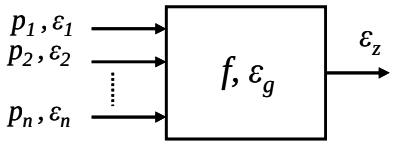

Figure 1.1 Gate implementing function f

First consider the error probability equation for a buffer gate shown in Figure 1.2. The error occurs at the output if (i) the input is erroneous and the gate is error free or (ii) the gate is erroneous and the input is error free. Therefore, assuming independent faults for the input and the gate, the output error probability for a buffer can be written as:

$$\varepsilon_z = \varepsilon_{in}(1-\varepsilon_g)+(1-\varepsilon_{in})\varepsilon_g = \varepsilon_g + (1-2\varepsilon_g)\varepsilon_{in} \tag{1.12}$$

where ε_{in} is the error probability at the input of the buffer. It can be seen from this equation that the output error probability for buffer is independent from the input signal probability. Note Equation1.12 can also be used to express the output error probability of an inverter gate.

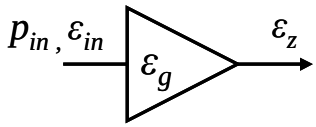

Figure 1.2 A faulty buffer with erroneous input

We can model each faulty gate with erroneous inputs as an ideal (no fault) gate with the same functionality and the same inputs in series with a faulty buffer as shown in Figure 1.3.

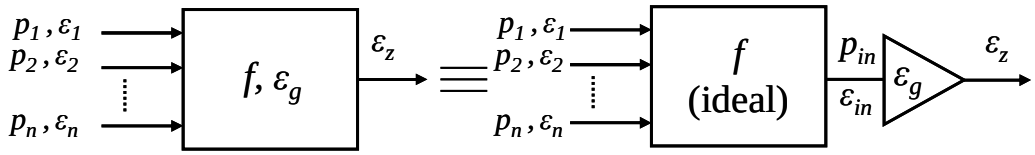

Figure 1.3 The proposed model for a general faulty gate

Now consider a general two-input gate. Using the fault model discussed above, we can write the output error probability considering all the cases of no error, single error and double errors at the input and the error in the gate itself. We can write the general equation for the error probability at the output, ε_z, as:

$$\varepsilon_z = \varepsilon_g + (1-2\varepsilon_g)\underbrace{\left(\begin{array}{l}\varepsilon_1(1-\varepsilon_2)\Pr\left\{\dfrac{\partial f}{\partial x_1}\right\}+(1-\varepsilon_1)\varepsilon_2\Pr\left\{\dfrac{\partial f}{\partial x_2}\right\} \\ +\varepsilon_1\varepsilon_2\Pr\left\{\dfrac{\Delta f}{\Delta(x_1x_2)}\right\}\end{array}\right)}_{\varepsilon_{in}} \tag{1.13}$$

where Pr{.} represents the signal probability function and returns the probability of its Boolean argument to be "1". The first and the second terms in ε_{in} account for the error at the output of the ideal gate due to single input errors at the first and the second inputs, respectively. Note error at each input of the ideal gate propagates to the output of this gate only if the other inputs are not masking it. The non-masking probability for each input error is taken into account by calculating the signal probability of the partial

Boolean difference of the function *f* with respect to the corresponding input. The first two terms in ε_{in} only account for the cases when we have single input errors at the input of the ideal gate, however, error can also occur when both inputs are erroneous simultaneously. This is taken into account by multiplying the probability of having simultaneous errors at both inputs, i.e., $\varepsilon_1\varepsilon_2$, with the probability of this error to be propagated to the output of the ideal gate, i.e., the signal probability of the total Boolean difference of *f* with respect to x_1x_2.
For 2-input AND gate ($f=x_1x_2$) shown in Figure 1.4 we have:

$$\begin{aligned}
&\Pr\left\{\frac{\partial f}{\partial x_1}\right\} = \Pr\{x_2\} = p_2\,,\ \Pr\left\{\frac{\partial f}{\partial x_2}\right\} = \Pr\{x_1\} = p_1 \\
&\Pr\left\{\frac{\Delta f}{\Delta(x_1x_2)}\right\} = \Pr\{\overline{x}_1\overline{x}_2 + x_1x_2\} = (1-p_1)(1-p_2) + p_1p_2 \\
&\qquad = 1-(p_1+p_2)+2p_1p_2
\end{aligned} \tag{1.14}$$

Plugging Equation 1.14 into Equation 1.13 and after some simplifications we have:

$$\varepsilon_{AND2} = \varepsilon_g + (1-2\varepsilon_g)\left(\varepsilon_1 p_2 + \varepsilon_2 p_1 + \varepsilon_1\varepsilon_2\left(1-2(p_1+p_2)+2p_1p_2\right)\right) \tag{1.15}$$

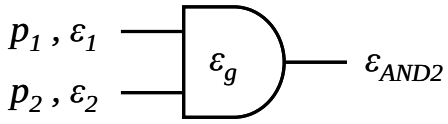

Figure 1.4 A 2-input faulty AND gate with erroneous inputs

Similarly, the error probability for the case of 2-input OR can be calculated as:

$$\varepsilon_{OR2} = \varepsilon_g + (1-2\varepsilon_g)\left(\varepsilon_1(1-p_2) + \varepsilon_2(1-p_1) + \varepsilon_1\varepsilon_2(2p_1p_2-1)\right) \tag{1.16}$$

And for 2-input XOR gate we have:

$$\varepsilon_{XOR2} = \varepsilon_g + (1-2\varepsilon_g)(\varepsilon_1 + \varepsilon_2 - 2\varepsilon_1\varepsilon_2) \tag{1.17}$$

It is interesting to note that the error probability at the output of the XOR gate is independent of the input signal probabilities. Generally, the 2-inpout XOR gate exhibits larger output error compared to 2-input OR and AND gates. This is expected since XOR gates show maximum sensitivity to input errors (XOR, like inversion, is an entropy-preserving function).The output error expression for a 3-input gate is:

$$\varepsilon_z = \varepsilon_g + (1-2\varepsilon_g)\left(\begin{array}{l}
\varepsilon_1(1-\varepsilon_2-\varepsilon_3+\varepsilon_2\varepsilon_3)\Pr\left\{\dfrac{\partial f}{\partial x_1}\right\} \\
+\varepsilon_2(1-\varepsilon_1-\varepsilon_3+\varepsilon_1\varepsilon_3)\Pr\left\{\dfrac{\partial f}{\partial x_2}\right\} \\
+\varepsilon_3(1-\varepsilon_1-\varepsilon_2+\varepsilon_1\varepsilon_2)\Pr\left\{\dfrac{\partial f}{\partial x_3}\right\} \\
+\varepsilon_1\varepsilon_2(1-\varepsilon_3)\Pr\left\{\dfrac{\Delta f}{\Delta(x_1x_2)}\right\} \\
+\varepsilon_2\varepsilon_3(1-\varepsilon_1)\Pr\left\{\dfrac{\Delta f}{\Delta(x_2x_3)}\right\} \\
+\varepsilon_1\varepsilon_3(1-\varepsilon_2)\Pr\left\{\dfrac{\Delta f}{\Delta(x_1x_3)}\right\} \\
+\varepsilon_1\varepsilon_2\varepsilon_3\Pr\left\{\dfrac{\Delta f}{\Delta(x_1x_2x_3)}\right\}
\end{array}\right) \tag{1.18}$$

As an example of a 3-input gate, we can use Equation 1.18 to calculate the probability of error for the case of 3-input AND gate. We can show that the output error probability can be calculated as:

$$\varepsilon_{AND3} = \varepsilon_g + (1-2\varepsilon_g)\left(\begin{array}{l}
p_2p_3\varepsilon_1 + p_1p_3\varepsilon_2 + p_1p_2\varepsilon_3 \\
+p_3(1-2(p_2+p_1)+2p_1p_2)\varepsilon_1\varepsilon_2 \\
+p_1(1-2(p_2+p_3)+2p_2p_3)\varepsilon_2\varepsilon_3 \\
+p_2(1-2(p_1+p_3)+2p_1p_3)\varepsilon_1\varepsilon_3 \\
+\left(\begin{array}{l}1-2(p_1+p_2+p_3) \\ +4(p_1p_2+p_2p_3+p_1p_3)-6p_1p_2p_3\end{array}\right)\varepsilon_1\varepsilon_2\varepsilon_3
\end{array}\right) \tag{1.19}$$

Now we give a general expression for a 4-input logic gate as:

$$\varepsilon_z = \varepsilon_g + (1-2\varepsilon_g)\left(\begin{array}{l} \sum_i \varepsilon_i \left(1 - \sum_{j\neq i}\varepsilon_j + \sum_{(j,k)\neq i}\varepsilon_j\varepsilon_k - \sum_{(j,k,l)\neq i}\varepsilon_j\varepsilon_k\varepsilon_l\right)\Pr\left\{\frac{\partial f}{\partial x_i}\right\} \\ +\sum_{(i,j)}\varepsilon_i\varepsilon_j\left(1 - \sum_{k\neq i,j}\varepsilon_k + \sum_{(k,l)\neq i,j}\varepsilon_k\varepsilon_l\right)\Pr\left\{\frac{\Delta f}{\Delta(x_i x_j)}\right\} \\ +\sum_{(i,j,k)}\varepsilon_i\varepsilon_j\varepsilon_k\left(1 - \sum_{l\neq i,j,k}\varepsilon_l\right)\Pr\left\{\frac{\Delta f}{\Delta(x_i x_j x_k)}\right\} \\ +\varepsilon_1\varepsilon_2\varepsilon_3\varepsilon_4\Pr\left\{\frac{\Delta f}{\Delta(x_1 x_2 x_3 x_4)}\right\} \end{array}\right) \tag{1.20}$$

The Boolean expression for a general k-input gate can be calculated in a similar manner.

1.3.2 Error Propagation in 2-to-1 Mux Using BDEC

We represent 2-to-1 Multiplexer (Mux) function as: $f = as + b\bar{s}$ .Using BDEC the output error probability in terms of the gate error probability, input signal probabilities and input error probabilities is:

$$\varepsilon_{Mux2to1} = \varepsilon_g + (1-2\varepsilon_g)\left(\begin{array}{l} \varepsilon_a(1-\varepsilon_b)(1-\varepsilon_s)\Pr\left(\frac{\partial f}{\partial a}\right) + \varepsilon_b(1-\varepsilon_a)(1-\varepsilon_s)\Pr\left(\frac{\partial f}{\partial b}\right) \\ +\varepsilon_s(1-\varepsilon_a)(1-\varepsilon_b)\Pr\left(\frac{\partial f}{\partial s}\right) + \varepsilon_a\varepsilon_b(1-\varepsilon_s)\Pr\left(\frac{\Delta f}{\Delta(ab)}\right) \\ +\varepsilon_a\varepsilon_s(1-\varepsilon_b)\Pr\left(\frac{\Delta f}{\Delta(as)}\right) + \varepsilon_b\varepsilon_s(1-\varepsilon_a)\Pr\left(\frac{\Delta f}{\Delta(bs)}\right) \\ +\varepsilon_a\varepsilon_b\varepsilon_s\Pr\left(\frac{\Delta f}{\Delta(abs)}\right) \end{array}\right) \tag{1.21}$$

Now we step wise show how to calculate various partial and total Boolean differences.

First we calculate all single variable partial Boolean differences as:

$$\frac{\partial f}{\partial a} = f_a \oplus f_{\bar{a}}$$
$$= \left(s + b\bar{s}\right) \oplus \left(b\bar{s}\right)$$
$$= \overline{\left(s + b\bar{s}\right)}.\left(b\bar{s}\right) + \left(s + b\bar{s}\right).\overline{\left(b\bar{s}\right)}$$
$$= \left(\bar{s}.\overline{b\bar{s}}\right).\left(b\bar{s}\right) + \left(s.\overline{b\bar{s}} + b\bar{s}.\overline{b\bar{s}}\right)$$
$$= \left(s.\overline{b\bar{s}}\right)$$
$$= s.\left(\bar{b} + s\right)$$
$$= s.\bar{b} + s$$
$$= s.\left(1 + \bar{b}\right)$$
$$= s$$

$$\frac{\partial f}{\partial b} = f_b \oplus f_{\bar{b}}$$
$$= \left(as + \bar{s}\right) \oplus \left(as\right)$$
$$= \overline{\left(as + \bar{s}\right)}.\left(as\right) + \left(as + \bar{s}\right).\overline{\left(as\right)}$$
$$= \left(\overline{as}.s\right).\left(as\right) + \left(as.\overline{as} + \bar{s}.\overline{as}\right)$$
$$= \left(\bar{s}.\overline{as}\right)$$
$$= \bar{s}.\left(\bar{a} + \bar{s}\right)$$
$$= \bar{a}\bar{s} + \bar{s}$$
$$= \bar{s}.\left(\bar{a} + 1\right)$$
$$= \bar{s}$$

$$\frac{\partial f}{\partial \left(s\right)} = f_s \oplus f_{\bar{s}}$$
$$= a \oplus b$$

Then we calculate two variables partial Boolean differences as:

$$\frac{\partial f}{\partial \left(ab\right)} = f_{ab} \oplus f_{\bar{a}\bar{b}}$$
$$= 1 \oplus 0$$
$$= 1$$

$$\frac{\partial f}{\partial \left(\bar{a}b\right)} = f_{\bar{a}b} \oplus f_{a\bar{b}}$$
$$= \bar{s} \oplus s$$
$$= 1$$

$$\frac{\partial f}{\partial \left(as\right)} = f_{as} \oplus f_{\bar{a}\bar{s}}$$
$$= 1 \oplus b$$
$$= \bar{b}$$

$$\frac{\partial f}{\partial \left(a\bar{s}\right)} = f_{a\bar{s}} \oplus f_{\bar{a}s}$$
$$= b \oplus 0$$
$$= b$$

$$\frac{\partial f}{\partial \left(bs\right)} = f_{bs} \oplus f_{\bar{b}\bar{s}}$$
$$= a \oplus 0$$
$$= a$$

$$\frac{\partial f}{\partial \left(b\bar{s}\right)} = f_{b\bar{s}} \oplus f_{\bar{b}s}$$
$$= 1 \oplus a$$
$$= \bar{a}$$

Finally we calculate three variable partial Boolean differences as:

$$\frac{\partial f}{\partial \left(abs\right)} = f_{abs} \oplus f_{\bar{a}\bar{b}\bar{s}}$$
$$= 1 \oplus 0$$
$$= 1$$

$$\frac{\partial f}{\partial \left(\bar{a}bs\right)} = f_{\bar{a}bs} \oplus f_{a\bar{b}\bar{s}}$$
$$= 0 \oplus 0$$
$$= 0$$

$$\frac{\partial f}{\partial \left(a\bar{b}s\right)} = f_{a\bar{b}s} \oplus f_{\bar{a}b\bar{s}}$$
$$= 1 \oplus 1$$
$$= 0$$

$$\frac{\partial f}{\partial \left(ab\bar{s}\right)} = f_{ab\bar{s}} \oplus f_{\bar{a}\bar{b}s}$$
$$= 1 \oplus 0$$
$$= 1$$

Next we calculate total Boolean differences as:

$$\frac{\Delta f}{\Delta \left(as\right)} = \frac{\partial f}{\partial a} \oplus \frac{\partial f}{\partial s} \oplus \frac{\partial^2 f}{\partial a \partial s}$$
$$= s \oplus \left(a \oplus b\right) \oplus 1$$
$$= s \oplus \overline{\left(a \oplus b\right)}$$
$$= s \oplus \left(a \otimes b\right)$$

$$\frac{\Delta f}{\Delta \left(bs\right)} = \frac{\partial f}{\partial b} \oplus \frac{\partial f}{\partial s} \oplus \frac{\partial^2 f}{\partial b \partial s}$$
$$= \bar{s} \oplus \left(a \oplus b\right) \oplus 1$$
$$= s \oplus \left(a \oplus b\right)$$

$$\frac{\Delta f}{\Delta \left(ab\right)} = \frac{\partial f}{\partial a} \oplus \frac{\partial f}{\partial b} \oplus \frac{\partial^2 f}{\partial a \partial b}$$
$$= s \oplus \bar{s} \oplus 0$$
$$= 1$$

$$\frac{\Delta f}{\Delta(abs)} = \frac{\partial f}{\partial a} \oplus \frac{\partial f}{\partial b} \oplus \frac{\partial f}{\partial s} \oplus \frac{\partial^2 f}{\partial a \partial s} \oplus \frac{\partial^2 f}{\partial b \partial s} \oplus \frac{\partial^2 f}{\partial a \partial b} \oplus \frac{\partial^3 f}{\partial a \partial b \partial s}$$
$$= s \oplus \bar{s} \oplus (a \oplus b) \oplus 1 \oplus 1 \oplus 0 \oplus 0$$
$$= s \oplus \bar{s} \oplus (a \oplus b)$$
$$= 1 \oplus (a \oplus b)$$
$$= a \otimes b$$

Plugging these values in Equation 1.21

$$\varepsilon_{Mux2to1} = \varepsilon_g + (1-2\varepsilon_g)\begin{pmatrix} \varepsilon_a(1-\varepsilon_b)(1-\varepsilon_s)p_s + \varepsilon_b(1-\varepsilon_a)(1-\varepsilon_s)(1-p_s) \\ +\varepsilon_s(1-\varepsilon_a)(1-\varepsilon_b)\Pr(a \oplus b) + \varepsilon_a\varepsilon_b(1-\varepsilon_s) \\ +\varepsilon_a\varepsilon_s(1-\varepsilon_b)\Pr(s \oplus (a \otimes b)) + \varepsilon_b\varepsilon_s(1-\varepsilon_a)\Pr(s \oplus (a \oplus b)) \\ +\varepsilon_a\varepsilon_b\varepsilon_s \Pr(a \otimes b) \end{pmatrix} \quad (1.22)$$

1.3.3 Circuit Error Model

In this section we use the gate error model proposed in sub-section 1.3.1 to calculate the error probability at the output of a given circuit. Given a multi-level logic circuit composed of logic gates, we start from the primary inputs and move toward the primary outputs by using a post-order (reverse DFS) traversal. For each gate, we calculate the output error probability using input signal probabilities, input error probabilities, and gate error probability and utilizing the error model proposed in sub-section 1.3.1. The signal probability for the output of each gate is also calculated based on the input signal probabilities and the gate function. The process of output error and signal probability calculation is continued until all the gates are processed. For each node *z* in the circuit, reliability is defined as:

$$\chi_z = 1 - \varepsilon_z \quad (1.23)$$

After processing all the gates in the circuit and calculating error probabilities and reliabilities for all the circuit primary outputs, we can calculate the overall circuit reliability. Assuming that different primary outputs of the circuit are independent, the overall circuit reliability can be calculated as the product of all the primary outputs reliabilities, that is:

$$\chi_{circuit} = \prod_i \chi_{PO_i} \quad (1.24)$$

The case of dependent primary outputs (which is obviously a more realistic scenario) requires calculation of spatial correlation coefficient as will be outlined

further on in the paper. The detailed treatment of spatial correlation coefficient calculation however falls outside the scope of the present work.

This error propagation algorithm has a complexity of $O(2^kN)$ where k is the maximum number of inputs to any gate in the circuit (which is small and can be upper bounded *a priori* in order to give $O(N)$ complexity) and N is the number of gates in the circuit. This complexity should be contrasted to that of the PTM based or the PDD-based approaches, that have a worst case complexity of $O(2^N)$. The tradeoff is that our proposed approach based on post-order traversal of the circuit netlist and application of Boolean difference operator results in only approximate output error and signal probability values due to the effect of re-convergent fanout structures in the circuit, which create *spatial correlations* among input signals to a gate. This problem has been extensively addressed in the literature on improving the accuracy of signal probability calculators [17][18]. Our future implementation of BDEC shall focus on utilizing similar techniques (including efficient calculation of spatial correlation coefficients) to improve the accuracy of proposed Boolean difference-based error calculation engine.

1.4 Practical Considerations

In this section we use the error models introduced in previous section to calculate the exact error probability expression at the output of a tree-structured circuit.

1.4.1 Output error expression

For the sake of elaboration, we choose a 4-input AND gate implemented as a balanced tree of 2-input AND gates as shown in Figure 1.5. We can calculate the output error of this circuit by expressing the error at the output of each gate using Equation 18.

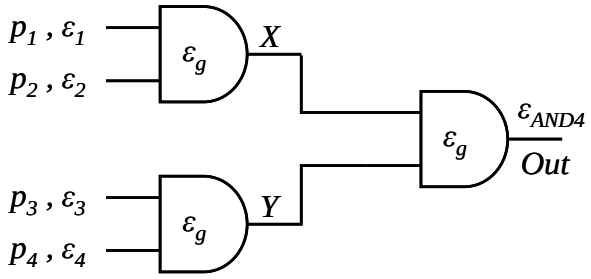

Figure 1.5 Balanced tree implementation of 4-input AND gate

Equation 1.25 provides the exact output error probability of the circuit shown in Figure 1.5 in terms of the input signal probabilities and input error probabilities where similar to [18] higher order exponents of the signal probabilities are reduced to first order exponents.

$$
\begin{aligned}
\varepsilon_{AND4} = {} & \left(p_2 p_3 p_4 \varepsilon_1 + p_1 p_3 p_4 \varepsilon_2 + p_1 p_2 p_4 \varepsilon_3 + p_1 p_2 p_3 \varepsilon_4 \right) \\
& + p_3 p_4 \left(1 - 2\left(p_1 + p_2\right) + 2 p_1 p_2\right) \varepsilon_1 \varepsilon_2 \\
& + p_2 p_4 \left(1 - 2\left(p_1 + p_3\right) + 2 p_1 p_3\right) \varepsilon_1 \varepsilon_3 \\
& + p_2 p_3 \left(1 - 2\left(p_1 + p_4\right) + 2 p_1 p_4\right) \varepsilon_1 \varepsilon_4 \\
& + p_1 p_4 \left(1 - 2\left(p_2 + p_3\right) + 2 p_2 p_3\right) \varepsilon_2 \varepsilon_3 \\
& + p_1 p_3 \left(1 - 2\left(p_2 + p_4\right) + 2 p_2 p_4\right) \varepsilon_2 \varepsilon_4 \\
& + p_1 p_2 \left(1 - 2\left(p_3 + p_4\right) + 2 p_3 p_4\right) \varepsilon_3 \varepsilon_4 \\
& + p_4 \left(1 - 2\left(p_1 + p_2 + p_3\right) + 4\left(p_1 p_2 + p_1 p_3 + p_2 p_3\right) - 6 p_1 p_2 p_3\right) \varepsilon_1 \varepsilon_2 \varepsilon_3 \\
& + p_2 \left(1 - 2\left(p_1 + p_3 + p_4\right) + 4\left(p_1 p_3 + p_1 p_4 + p_3 p_4\right) - 6 p_1 p_3 p_4\right) \varepsilon_1 \varepsilon_3 \varepsilon_4 \\
& + p_3 \left(1 - 2\left(p_1 + p_2 + p_4\right) + 4\left(p_1 p_2 + p_1 p_4 + p_2 p_4\right) - 6 p_1 p_2 p_4\right) \varepsilon_1 \varepsilon_2 \varepsilon_4 \\
& + p_1 \left(1 - 2\left(p_2 + p_3 + p_4\right) + 4\left(p_2 p_3 + p_2 p_4 + p_3 p_4\right) - 6 p_2 p_3 p_4\right) \varepsilon_2 \varepsilon_3 \varepsilon_4 \\
& + \begin{pmatrix} 1 - 2\left(p_1 + p_2 + p_3 + p_4\right) \\ +4\left(p_1 p_2 + p_1 p_3 + p_1 p_4 + p_2 p_3 + p_2 p_4 + p_3 p_4\right) \\ -8\left(p_1 p_2 p_3 + p_1 p_2 p_4 + p_1 p_3 p_4 + p_2 p_3 p_4\right) \\ +14 p_1 p_2 p_3 p_4 \end{pmatrix} \varepsilon_1 \varepsilon_2 \varepsilon_3 \varepsilon_4
\end{aligned}
\tag{1.25}
$$

In Equation 1.25, without loss of generality, we assume $\varepsilon_{g=0}$ in order to reduce the length of the expression.

Using symbolic notation along with higher order exponent suppression, the model presented in section 1.3 can compute the exact output error probability in circuits with no reconvergent fanout. We will show in next section that by sacrificing little accuracy and using numerical values instead of symbolic notation, the computational complexity of our gate error model becomes linear in terms of the number of gates in the circuit.

1.4.2 Reconvergent Fanout

Figure 1.6 shows an example of a circuit with reconvergent fanout. It is clear from the figure that inputs to the final logic gate are not independent. Therefore, if the BDEC technique discussed in Section 1.3 is applied to this circuit, the calculated output error probability will not be accurate. In this section we describe a modification to the BDEC technique that improves the probability of error for the circuit in the presence of reconvergent fanout structures in the circuit.

Local reconvergent fanout such as the one depicted in Figure 1.6 can be handled by collapsing levels of logic. For this example, we consider all the four gates in Figure 1.6 as a single super gate and then apply the BDEC technique to this super gate. For the input to output error propagation in the original circuit, BDEC will ignore the internal structure of the super gate and only considers the actual function implemented by the

super gate, 2-to-1 Mux in this case. The original implementation information can be taken into account by properly calculating the ε_g value for this new 3-input super gate.

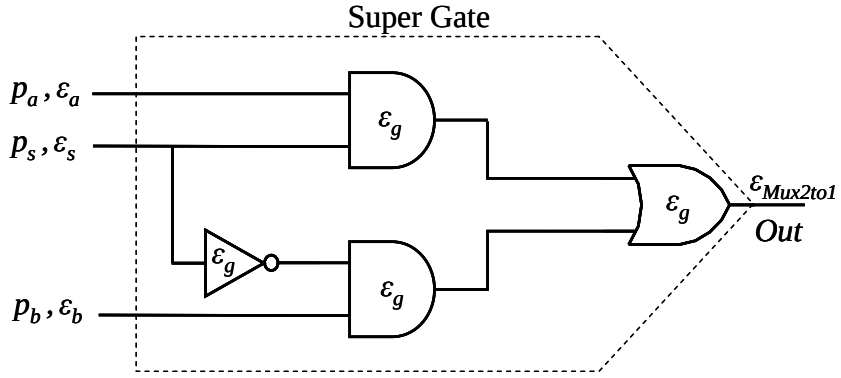

Figure 1.6 Re-convergent fanout in a 2-to-1 Multiplexer

The ε_g value for the collapsed gate is calculated using BDEC for the original circuit block before collapsing but assuming that the input error probabilities are zero. For example for the circuit in Figure 1.6 the error probability at the output of the top AND gate and the inverter using BDEC equations described in section 1.3.1 and assuming input error probabilities to be zero will be ε_g each. Similarly the error probability at the output of the bottom AND gate will be $\varepsilon_{AND2} = \varepsilon_g + (1-2\varepsilon_g)(\varepsilon_g p_b)$. Likewise we can calculate the expression for the error probability at the output of the OR gate which in this case will be the ε_g of the super gate. Equation 1.26 shows the final expression for the ε_g value of the collapsed gate; note that the ε_g value for the collapsed gate is also a function of the input signal probabilities. As discussed in Section 1.4.1, the error expression for the super gate ε_g has been obtained after suppressing the exponents of signal probabilities that are greater than "1" to "1". In contrast we do not suppress the exponents of ε_g values since this higher exponent may have correctly resulted from the fact that each gate in the circuit can fail with same error probability On the other hand the higher exponent may have arisen from the fact that the error of some multiple-fanout gate is propagated to a reconvergent fanout point through different paths in the circuit, and hence, the higher exponent must indeed be suppressed. So there is some inaccuracy in our proposed method. To be able to decide precisely whether or not the exponents of ε_g must be suppressed, we will have to use a unique symbol for each gate's error probability and propagate these unique symbols throughout the circuit while suppressing the higher exponents of each unique symbol. The results reported in Table 1.1 have been obtained using the estimation of super gate's ε_g from our implementation of BDEC in SIS [20], which does not include exponent suppression.

$$\begin{aligned}\varepsilon_g^* &= 3\varepsilon_g - 5\varepsilon_g^2 + 2\varepsilon_g^3 \\ &+ p_b p_s \varepsilon_g - p_a p_s \varepsilon_g - p_a p_b p_s \varepsilon_g \\ &- 3p_b \varepsilon_g^2 + 2p_a p_s \varepsilon_g^2 - 2p_b p_s \varepsilon_g^2 + 4p_a p_b p_s \varepsilon_g^2 \\ &+ 8p_b \varepsilon_g^3 - 4p_a p_b p_s \varepsilon_g^3 - 4p_b \varepsilon_g^4\end{aligned} \tag{1.26}$$

Table 1.1 Output error probability with re-convergent fanout

S/N	Pa	Pb	Ps	ε_g	ε_a	ε_b	ε_s	BDEC	BDEC-CLP	PTM
1	0.5	0.5	0.5	0.05	0.05	0.05	0.05	0.1814	0.1835	0.1829
2	0.1	0.2	0.3	0.05	0.05	0.05	0.05	0.1689	0.1820	0.1852
3	0.5	0.6	0.7	0.05	0.05	0.05	0.05	0.1909	0.1827	0.1830
4	0.7	0.8	0.9	0.05	0.05	0.05	0.05	0.1858	0.1684	0.1668
5	0.5	0.5	0.5	0.001	0.001	0.001	0.001	0.0044	0.0044	0.0044
6	0.5	0.5	0.5	0.002	0.001	0.001	0.001	0.0072	0.0072	0.0072
7	0.5	0.5	0.5	0.01	0.01	0.01	0.01	0.0421	0.0422	0.0422
8	0.5	0.5	0.5	0.02	0.01	0.02	0.03	0.0842	0.0815	0.0814
9	0.5	0.5	0.5	0.08	0.08	0.08	0.08	0.2603	0.2645	0.2633
10	0.5	0.5	0.5	0.05	0.06	0.07	0.08	0.2027	0.2037	0.2032

Table 1.1 shows that BDEC + logic collapsing produces accurate results for the circuit in Figure 1.6. Table 1.2 shows the comparison of percent error for BDEC and BDEC + collapsing as compared to PTM. If the reconvergent fanout extends over multiple circuit levels then multiple level collapsing can be used but after few levels, the computational complexity of computing output error probability of a super gate with many inputs will become prohibitive and a trade-off between accuracy and complexity will have to be made.

Table 1.2 Percent error reduction in output error probability using BDEC +Collapsing

S/N	Pa	Pb	Ps	ε_g	ε_a	ε_b	ε_s	BDEC Error	BDEC-CLP Error
1	0.5	0.5	0.5	0.05	0.05	0.05	0.05	0.84%	0.33%
2	0.1	0.2	0.3	0.05	0.05	0.05	0.05	8.78%	1.75%
3	0.5	0.6	0.7	0.05	0.05	0.05	0.05	4.34%	0.15%
4	0.7	0.8	0.9	0.05	0.05	0.05	0.05	11.39%	0.94%
5	0.5	0.5	0.5	0.001	0.001	0.001	0.001	0.02%	0.01%
6	0.5	0.5	0.5	0.002	0.001	0.001	0.001	0.01%	0.01%
7	0.5	0.5	0.5	0.01	0.01	0.01	0.01	0.21%	0.08%
8	0.5	0.5	0.5	0.02	0.01	0.02	0.03	3.48%	0.12%
9	0.5	0.5	0.5	0.08	0.08	0.08	0.08	1.13%	0.46%
10	0.5	0.5	0.5	0.05	0.06	0.07	0.08	0.25%	0.24%
11	-	-	-	-	-	-	**Average**	3.05%	0.41%

In passing, we point out that the correlation coefficient method and partial collapse methods both tackle the same problem, that is, how to account for the correlations due to reconvergent fanout structures in VLSI circuits. The tradeoff is that the correlation coefficient method has high complexity due to the requirement to calculate and propagate all correlation coefficients along with signal and error probabilities whereas the partial collapse has high complexity due to the need to calculate output signals and error probabilities of super gates with a large number of inputs. In practice, the partial collapse of 2 or 3 levels of logic into each node (super gate) or the computation of only pair wise spatial correlations is adequate and provides high accuracy.

1.5 Simulation Results

In this section we present some simulation results for the proposed circuit reliability technique and we compare the results of our approach with those of PTM and PGM [19].

We implemented the proposed error calculator and algorithm (BDEC) in SIS [20]. SIS has been widely used by logic synthesis community for designing combinational and sequential logic circuits. We extended existing logic simulation in SIS with faulty circuit simulation based on Monte Carlo simulation technique. We attached a probability function with each node which flips the correct output of the node with a predefined error probability. We used this Monte Carlo simulation to form a reference to compare BDEC results for medium and large circuits.

We added a new BDEC module to the existing SIS package. While simulating a logic circuit, BDEC module models each gate as a probabilistic gate. We used the built-in co-factor function in SIS to develop partial Boolean difference and total Boolean difference functions that are used to propagate single and simultaneous multiple errors from the inputs to the output of the gate respectively. We have also implemented level collapsing to overcome the inaccuracies introduced because of local reconvergent fanouts. Note that while collapsing levels of logic, we do not change the original logic network; instead, we simply recalculate and update the error and signal probability at the output of the nodes that have reconvergent fanout structures inside their corresponding super gate.

In the past, SIS has been used to apply various delay, area and power level optimizations to logic circuits. By incorporating BDEC module to SIS, we expect that researches will be able to use SIS to develop reliability-aware optimizations for logic circuits. For example, given a library of gates with different levels of reliability, design a circuit with given functionality that minimizes area, delay and power overheads while meeting a given reliability constraint.

Regarding simulation results in this section, for simplicity, but without loss of generality, we assume all gates in a circuit have the same gate error probability ε_g. All primary inputs are assumed to be error free and spatiotemporally uncorrelated. Moreover, signal probability for all the inputs were set to 0.5. The gate error probability was set to 0.05. We thus present results that show how efficiently BDEC can calculate the output reliability for circuits with high primary input count. Running our MATLAB

7.1-based implementation of PTM on a computer system with 2GBytes of RAM, we observed that typically for circuits with 16 or more inputs, PTM reported out of memory error. BDEC, however, does the calculations much faster and more efficient than PTM.

Table 1.3 shows the results for reliability calculation for some tree-structured circuits. For example, "8-Input XOR BT" (BT for Balanced Tree) refers to 8-input XOR function implemented using 2-input XOR gates in three levels of logic whereas "8-Input XOR Chain" refers to the same function realized as a linear chain of seven 2-input XOR gates. We also show results for two 16-input circuits with balanced tree implementation of 2-input gates having layers of 2-input AND, OR or XOR gates. First letter of gate name is used to show the gates used in each level. For example, AOXO means that the circuits consists of four levels of logic with AND, OR, XOR and OR gates at the first, second, third and fourth level, respectively. Since the complexity of the PTM approach increases with the number of primary inputs exponentially, all the circuits in Table 1.3 are chosen to have relatively small number of primary inputs. Second and third columns of this table compare the execution times for PTM and BDEC, respectively, while the forth and the fifth columns compare the output reliability for the two approaches. It can be seen that our proposed BDEC technique achieves highly accurate reliability values, i.e., the reliability values are different than PTM ones by at most 0.1% for the circuits reported in Table 1.3 . More importantly, Table 1.3 shows the difference between the scaling trend of the execution time in both PTM and BDEC techniques. In PTM, the execution time increases exponentially when we move from smaller circuits to larger circuits in Table 1.3, whereas in BDEC the change in the execution time when we move from smaller circuits to the larger ones in Table 1.3 is really small. For two cases, 16-input XOR chain and 16-input AND chain, the system runs out of memory while executing PTM technique. This shows that execution of PTM technique for even relatively small circuits needs a huge amount of system memory.

Table 1.3 Circuit reliability for tree-structured circuits having relatively small number of PIs

Benchmarks	**# of Gates**	**Execution Time (ms)**		**Circuit Reliability**	
		PTM	**BDEC**	**PTM**	**BDEC**
8-Input XOR BT	7	0.790	0.011	0.7391	0.7391
16-Input XOR BT	15	1664.5	0.017	0.6029	0.6029
16-Input XOR Chain	15	Out of Memory	0.015	Out of Memory	0.6029
8-Input AND BT	7	0.794	0.010	0.9392	0.9382
16-Input AND BT	15	1752.2	0.017	0.9465	0.9462
16-Input AND Chain	15	Out of Memory	0.016	Out of Memory	0.9091
16-input AOXO BT	15	1769.3	0.017	0.7622	0.7616
16-input OXAX BT	15	1593.1	0.017	0.7361	0.7361

Another important advantage of the proposed BDEC technique which can be observed from Table 1.3 is that the complexity of this technique mainly depends on the number of the gates in the circuit; however, the complexity of PTM technique depends on several other factors such as number of the inputs, width and depth of the circuit, number of the wire crossovers, etc. In other words, efficiency (execution time and memory usage) of PTM depends not only on the number of the gates in the circuit, but

on the circuit topology. This is a big disadvantage for PTM making it an infeasible solution for large and/or topologically complex circuits.

It is worth mentioning that although the complexity of Boolean difference equations increases exponentially with the number of the inputs of the function; this does not increase the complexity of the BDEC technique. The reason is the fact that using gates with more than few inputs, say 4, in the actual implementation of any Boolean function is not considered as a good design practice. This makes the complexity of calculating Boolean difference equations small. On the other hand for a fixed library of gates, all the Boolean difference equations can be calculated offline, so, there is no computational overhead due to calculating the Boolean difference equations in BDEC.

Table 1.4 Circuit Reliability for Tree-Structured Circuits having relatively Large Number of PIs

Circuit	# of Gates	Execution Time (ms)	Circuit Reliability
64-Input XOR (BT)	63	0.046	0.5007
64-Input XOR (Chain)	63	0.043	0.5007
64-Input AND (BT)	63	0.054	0.9475
64-Input AND (Chain)	63	0.051	0.9091
64-Input AOXAOXBT	63	0.054	0.6314
64-Input XAOXAOBT	63	0.053	0.9475
16-Bit RCA	80	0.115	0.0132
32-Bit RCA	160	0.216	0.0002
I1	46	0.054	0.3580
C18	6	0.013	0.8032

Table 1.4 shows the results, execution time and reliability calculation for some of synthesized tree-structured circuits with relatively larger number of inputs. Since the complexity of the PTM is really high for these circuits we only show the results for BDEC. Some of the circuits in Table 1.4 are the larger versions of the circuits reported in Table 1.3. We have also included 16 and 32-bit ripple carry adder (RCA) circuits. Results for two benchmark circuits, I1 and C18, are also included in this table.

From the results of Table 1.3 and Table 1.4 we note that circuits that use more XOR gates will incur smaller output reliability under a uniform gate failure probability. Furthermore, moving XOR gates closer to the primary outputs results in lower output reliability. Therefore, in order to have more reliable designs, we must have lower XOR gates close to the primary outputs.

Table 1.5 Circuit Reliability and Efficiency of BDEC Compared to PGM and PTM

Circuit	Execution Time (ms)		Circuit Reliability (ε_g=0.05)			% Error Compared to PTM	
	BDEC	PTM	BDEC	PGM	PTM	BDEC	PGM
2-4 Decoder	0.014	6.726	0.7410	0.7397	0.7479	0.92%	1.10%
FA1	0.013	2.399	0.7875	0.7898	0.8099	2.77%	2.48%
FA2	0.017	3.318	0.6326	0.5933	0.6533	3.17%	9.18%
C17	0.012	2.304	0.7636	0.7620	0.7839	2.59%	2.79%
Comp.	0.014	0.937	0.7511	0.7292	0.8264	9.11%	11.76%
Avg. Err.	-	-	-	-	-	3.71%	5.46%

Table 1.5 compares the results for PTM, PGM [19], and BDEC for some more general circuits. Note FA1 and FA2 are two different implementations of full adder circuit. The former is XOR/AND implementation and the latter is NAND only implementation. Also Comp. is a two-bit comparator circuit. We report the results for our implementation of PTM and BDEC; however, since we were not able to produce the results of PGM, we took the reported results in [19]. As it can be seen from this table, BDEC shows better accuracy as compared to PGM.

Table 1.6 Runtime Comparison between BDEC and PTM for some Large Benchmark Circuits

Benchmark	# of Gates	PIs	POs	BDEC Exec Time (sec)	PTM Exec Times (sec)
C17	6	5	2	7.00E-06	0.313
Pcle	71	19	9	2.40E-05	4.300
z4ml	74	7	4	2.20E-05	0.840
Mux	106	21	1	2.80E-05	2.113
9symml	252	9	1	5.20E-05	696.211

Table 1.6 shows the results of running BDEC for somewhat larger benchmark circuits. In the last column, we report the results for some of the circuits that were analyzed in [5] to compare the run times of running PTM with that of BDEC. PTM results were reported for technology independent benchmarks where as BDEC results are for benchmark circuits mapped to a cell library in 65nm CMOS technology. PTM results were generated using a system with 3GHz Pentium 4 processor where as BDEC results are generated from a system with 2.4GHz dual core processor. One can see that BDEC (which has very low memory usage) is orders of magnitude faster than PTM.

Table 1.7 shows how BDEC execution times linearly scale with the number of gates. As it was mentioned in the introduction of this chapter, the worst-case time complexity of previously proposed techniques such as PTM and PDD is exponential in terms of the number of the gates in the circuit.

Table 1.7 Circuit Reliability for Large Benchmark Circuits

Benchmark	# of Gates	PIs	POs	BDEC Exec Time (μ sec)	BDEC Reported Reliability
Majority	22	5	1	9.0	0.6994
Decod	66	5	16	18.0	0.2120
Count	139	31	16	38.0	0.0707
frg1	143	28	3	48.0	0.6135
C880	442	60	26	96.0	0.0038
C3540	1549	50	22	358.0	0.0003
alu4	2492	12	8	577.0	0.0232
t481	4767	16	1	1710.0	0.8630

Table 1.8 shows how BDEC execution times and reliability calculations compared to those of Monte Carlo (MC) simulations. We could not run PTM for larger circuits because of out of bound memory requirements to store probability transfer matrices hence we resorted to MC simulations. In most of the cases we ran 10000 iterations of MC simulations where each input changed with the probability of 0.5. In the case of higher input count we ran up to 1M iterations to get more accurate results, but the

execution times reported in 4th column of Table 1.8 are for 10000 iterations in each case. Since overall circuit reliability for multi output circuits tend to be very low we also report BDEC calculated minimum output reliability for single output in the last column of Table 1.8.

Table 1.8 BDEC Circuit Reliability Compared to MC Simulations for Large Benchmark Circuits

Benchmark	# of Gates	POs	MC Exec Time (sec)	BDEC Exec Time (μ sec)	MC Reported Reliability	BDEC Reported Reliability	% Error	Min Single Output Reliability
majority	22	1	0.25	2244	0.6616	0.6994	5.71	0.6994
decod	66	16	0.69	6234	0.2159	0.2120	1.81	0.8820
pcle	71	9	0.82	6899	0.2245	0.2270	1.11	0.8401
cordic	116	2	1.26	10093	0.5443	0.5299	2.65	0.7220
sct	143	15	1.54	13086	0.1310	0.130	0.76	0.7988
frg1	143	3	1.59	13864	0.5895	0.6135	4.07	0.7822
b9	147	21	1.64	14118	0.0271	0.0261	3.69	0.7223
lal	179	19	2.52	18001	0.0924	0.0990	7.14	0.8067
9symml	252	1	2.90	27225	0.7410	0.6189	16.48	0.6189
9sym	429	1	4.93	48039	0.7705	0.6398	16.96	0.6398
C5315	2516	123	33.34	267793	0.0000	0.0000	0.00	0.5822
Average	-	-	-	-	-	-	5.49	-

1.6 Extensions to BDEC

1.6.1 Soft Error Rate (SER) estimation using BDEC

As technology scales down, the node-level capacitance (which is a measure of the stored charge at the output of the node) and the supply voltage decrease, hence, soft error rates are increasing exponentially [23]. Soft errors in CMOS ICs are caused by a particle (Alpha, energetic neutron, etc.) striking a node which is holding some data value. Soft errors in general result in discharging of a node capacitance which in a combinational circuit means a “1” to “0” transition. This type of error is thus different from Von Neumann error discussed so far in the paper. A soft error in SRAM can change the logic value stored in the SRAM and thus, can be thought as a flipping error.

To use BDEC for soft error rate estimation of combinational logic circuits, we modify the BDEC equations developed in Section 1.3.1. We still use the Boolean Difference Calculus method to find out the conditions when an error on one or more inputs will affect the output of the gate. We also assume a sufficiently large latching window for a soft error so that such an error can in the worst case propagate to the primary output(s) of the target combinational circuit. In the following, we show the equations to calculate the soft error rate at the output of a buffer, a 2-input AND gate and a 2-input XOR gate. Note $\varepsilon_{g,\ soft}$ in the following equations means the probability that a soft error at the output of the gate will cause the output to transition from logic “1” to logic “0”.

To calculate the soft error rate expression at the output of a buffer, we note that soft error happens only when the input is "1" and either of the input or output is affected. That is:

$$\varepsilon_{buf,soft} = p_{in}\left(\varepsilon_{in,soft} + \varepsilon_{g,soft} - \varepsilon_{in,soft}\varepsilon_{g,soft}\right) \tag{1.27}$$

where $\varepsilon_{in,\ soft}$ is the soft error rate at the input, and the term in the parentheses is the probability of error at the input or the output.

To calculate the soft error rate at the output of a 2-input AND gate, we pay attention to the truth table of this gate knowing that soft error can only make "1" to "0" changes. This leads us to the fact that the only time that the output value of a 2-input AND gate is affected by a soft error is when both inputs are "1" and an error occurs at any of the inputs or at the output. Therefore, the soft error rate at the output of a 2-input AND gate is written as:

$$\varepsilon_{AND2,soft} = p_1 p_2 \begin{pmatrix} \varepsilon_{1,soft} + \varepsilon_{2,soft} + \varepsilon_{g,soft} \\ -\varepsilon_{1,soft}\varepsilon_{2,soft} - \varepsilon_{1,soft}\varepsilon_{g,soft} - \varepsilon_{2,soft}\varepsilon_{g,soft} \\ +\varepsilon_{1,soft}\varepsilon_{2,soft}\varepsilon_{g,soft} \end{pmatrix} \tag{1.28}$$

Similarly, the soft error rate at the output of a 2-input XOR gate can be calculated by looking into its truth table and realizing that the output value can be affected by a soft error when: (i) exactly one input is "1" and one input is "0" and soft error changes the logic-1 input or the output, or (ii) both inputs are "1" and soft error changes one and only one of these logic-1 inputs. Therefore, the soft error rate at the output of a 2-input XOR gate is calculated as:

$$\begin{aligned} \varepsilon_{XOR2,soft} = {} & p_1(1-p_2)\left(\varepsilon_{1,soft} + \varepsilon_{g,soft} - \varepsilon_{1,soft}\varepsilon_{g,soft}\right) \\ & + (1-p_1)p_2\left(\varepsilon_{2,soft} + \varepsilon_{g,soft} - \varepsilon_{2,soft}\varepsilon_{g,soft}\right) \\ & + p_1 p_2\left(\varepsilon_{1,soft}\left(1-\varepsilon_{2,soft}\right) + \left(1-\varepsilon_{1,soft}\right)\varepsilon_{2,soft}\right) \end{aligned} \tag{1.29}$$

Similarly we can derive error equations for other types of gate functions.

1.6.2 BDEC for asymmetric erroneous transition probabilities

BDEC for Von Neumann fault model assumed equal probability of error for a "0" to "1" and "1" to "0" erroneous transition. But this may not always be the case for example in dynamic and domino logic families, the only possible erroneous transition during the evaluate mode is from "1" to "0". In these situations, the solution is to independently calculate the overall circuit error probability using the low-to-high and high-to-low probability values and gate-level error rates. Both circuit error rates are then reported.

1.6.3 BDEC applied to Emerging Nano Technologies

A quantum-dot cellular automaton (QCA) [21] is a binary logic architecture which can miniaturize digital circuits to the molecular levels and operate at very low power levels [22]. QCA devices encode and process binary information as charge configurations in arrays of coupled quantum dots, rather than current and voltage levels. One unique aspect of QCA is that both wires and gates are constructed from quantum dots. Each dot consists of a pair of electrons that can be configured in two different ways to represent a single bit of information. Hence in QCA both gates and wire are subject to bit-flip errors. QCAs have two main sources of error: 1) decay (decoherence)—when electrons that store information are lost to the environment, and 2) switching error—when the electrons do not properly switch from one state to another due to background noise or voltage fluctuations [22]. BDEC uses Von Neumann (Bit-flip) fault model, hence it is thus well suited to calculate errors in QCAs. In QCA wires/interconnects can also make bit-flip errors, hence BDEC must be extended to be used for QCAs. This extension in BDEC is straightforward and requires a simple replacement of each interconnect in the circuit with a probabilistically faulty buffer.

1.7 Conclusions

As technology scales down circuit reliability is becoming one of the main concerns in VLSI design. In nano scale CMOS regime circuit reliability have to be considered in the early design phases. This shows the need for fast reliability calculator tools that are accurate enough to estimate overall circuit reliability. The presented error/reliability calculator, BDEC, takes primary input signal and error probabilities and gate error probabilities and computes the reliability of the circuit. BDEC benefits from a linear-time complexity with number of the gates in the circuit. Compared to PTM which generates accurate reliability results, BDEC generates highly accurate results that are very close to PTM ones. We showed that the efficiency, execution time and memory usage, of BDEC is much better than those for PTM.

BDEC can find application in any combinatorial logic design where reliability is a major concern. Presently BDEC can be applied to combinatorial circuits only, sequential logic is not supported. BDEC can be easily enhanced to be applied to sequential logic. Current version of BDEC uses level collapsing to reduce the effect of re-convergent fanout. In future BDEC can be enhanced to use spatial correlations between the signals to further reduce the inaccuracies introduced because of re-convergent fanouts.

Bibliography

[1] Krishnaswamy Smita (2008) Design, Analysis, and Test of Logic Circuits under Uncertainty, Dissertation, University of Michigan at Ann Arbor (USA)

[2] Rabey J M, Chankrakasan A, Nikolic B: Digital Integrated Circuits, Prentice Hall, 2003. pp. 445-490.

[3] Hu C (1999) Silicon nanoelectronics for the 21st century Nanotechnology. vol. 10, No. 2, Page(s): 113-116.

[4] Bahar R I , Lau C, Hammerstrom D, Marculescu D, Harlow J, Orailoglu A, Joyner W H Jr., Pedram M (2007) Architectures for silicon nanoelectronics and beyond. Computer, v 40, n 1, Page(s): 25-33.

[5] Krishnaswamy S, Viamontes G F, Markov I L , Hayes J P (2005) Accurate reliability evaluation and enhancement via probabilistic transfer matrices. Proceedongs of Design, Automation and Test in Europe (DATE), Page(s):282-287.

[6] Abdollahi A (2007) Probabilistic Decision Diagrams for Exact Probabilistic Analysis. Proc. International Conference on Computer Aided Design (ICCAD)

[7] Mehta H, Borah M, Owens R M, Irwin M J (1995) Accurate estimation of combinational circuit activity. Proc. Design Automation Conference, (DAC) Page(s): 618-622.

[8] Bhaduri D and Shukla S (2004) NANOPRISM: A tool for evaluating granularity versus reliability trade-offs in nano architectures. in Proc. 14th ACM Great Lakes Symp. VLSI, Page(s): 109–112.

[9] Norman G, Parker D, Kwiatkowska M, and Shukla S (2005) Evaluating the reliability of NAND multiplexing with PRISM. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 24, no. 10, Page(s): 1629–1637

[10] Bhaduri D, Shukla S K, Graham P S, Gokhale M B (2007) Reliability Analysis of Large Circuits Using Scalable Techniques and Tools. IEEE Trans on Circuits and Systems, Volume 54, Issue 11, Page(s): 2447 – 2460.

[11] Sellers F F, Hsiao M Y, and Bearnson L W (1968) Analyzing Errors with the Boolean Difference," IEEE Transactions on Computers, vol. 17, No. 7, Page(s):676-683.

[12] Akers S B Jr (1959) On the theory of Boolean functions. SIAM J. Appl. Math., vol. 7, Page(s):487-498.

[13] Ku C T, and Masson G M (1975) The Boolean Difference and Multiple Fault Analysis. IEEE Trans. on Computers, vol. c-24, Page(s):62-71.

[14] Das S R, Srimani P K, and Dutta C R (1976) On Multiple Fault Analysis in Combinational Circuits by Means of Boolean Difference. Proc. of the IEEE, vol. 64, No. 9, pp. 1447-1449.

[15] Rejimon T, Bhanja S (2005) An accurate probabilistic model for error detection. Proc. 18th International Conference on VLSI Design, Page(s):717-722.

[16] Ercolani S, Favalli M, Damiani M, Olivo P, and Ricco B (1992) Testability Measures in Pseudorandom Testing. IEEE Trans. on CAD, vol. 11, Page(s):794-800.

[17] Marculescu R, Marculescu D and Pedram M (1998) Probabilistic modeling of dependencies during switching activity analysis. IEEE Trans. on Computer Aided Design, Page(s):73-83.

[18] Parker K P, McCluskey E J (1975) Probabilistic Treatment of General Combinational Networks. IEEE Trans on Computers,Volume 24 , Issue 6, Pages 668-670.

[19] Han J, Gao J B, Jonker P, Qi Yan and Fortes J A B (2005) Toward hardware-Redundant Fault-Tolerant Logic for Nanoelectronics. IEEE Trans on Design and Test of Computers, vol. 22-4 Page(s):328–339.

[20] Sentovich E M, Singh K J, Lavagno L, Moon C, Murgai R, Saldanha A, Savoj H, Stephan P R, Brayton R K, and SangiovanniVincentelli A (1992) SIS: A system for sequential circuit synthesis," U.C. Berkeley, Tech. Rep.

[21] Lent C S, Tougaw P D, Porod W, and Bernstein G H (1993) Quantum cellular automata. Nanotechnology, vol. 4, pp. 49–57

[22] Rejimon T, Bhanja S (2006) Probabilistic Error Model for Unreliable Nano-logic Gates. Proc. NANO, pp. 47-50

[23] Li L, Degalahal V, Vijaykrishnan N, Kandemir M, Irwin M J (2004) Soft error and energy consumption interactions: a data cache perspective. Proceedings of the international symposium on Low power electronics and design ISPLD'04

Index