Probabilistic Transfer Matrices in Symbolic Reliability Analysis of Logic Circuits

SMITA KRISHNASWAMY, GEORGE F. VIAMONTES, IGOR L. MARKOV, and JOHN P. HAYES
University of Michigan, Ann Arbor

We propose the probabilistic transfer matrix (PTM) framework to capture nondeterministic behavior in logic circuits. PTMs provide a concise description of both normal and faulty behavior, and are well-suited to reliability and error susceptibility calculations. A few simple composition rules based on connectivity can be used to recursively build larger PTMs (representing entire logic circuits) from smaller gate PTMs. PTMs for gates in series are combined using matrix multiplication, and PTMs for gates in parallel are combined using the tensor product operation. PTMs can accurately calculate joint output probabilities in the presence of reconvergent fanout and inseparable joint input distributions. To improve computational efficiency, we encode PTMs as algebraic decision diagrams (ADDs). We also develop equivalent ADD algorithms for newly defined matrix operations such as *eliminate_variables* and *eliminate_redundant_variables*, which aid in the numerical computation of circuit PTMs. We use PTMs to evaluate circuit reliability and derive polynomial approximations for circuit error probabilities in terms of gate error probabilities. PTMs can also analyze the effects of logic and electrical masking on error mitigation. We show that ignoring logic masking can overestimate errors by an order of magnitude. We incorporate electrical masking by computing error attenuation probabilities, based on analytical models, into an extended PTM framework for reliability computation. We further define a susceptibility measure to identify gates whose errors are not well masked. We show that hardening a few gates can significantly improve circuit reliability.

Categories and Subject Descriptors: B.6.2 [**Logic Design**]: Design Aids; B.6.3 [**Logic Design**]: Reliability and Testing

General Terms: Reliability, Performance

Additional Key Words and Phrases: Symbolic analysis, fault tolerance

ACM Reference Format:

Krishnaswamy, S., Viamontes, G. F., Markov, I. L., and Hayes, J. P. 2008. Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. ACM Trans. Des. Autom. Electron. Syst. 13, 1, Article 8 (January 2008), 35 pages. DOI = 10.1145/1297666.1297674 http://doi.acm.org/10.1145/1297666.1297674

This work has been supported by the National Science Foundation under Grant CCF-0205288, by the DARPA QuIST program, and by the U.S. Air Force under agreement No. FA8750-05-1-0282.
Authors' addresses: Department of Electrical Engineering and Computer Science, Advanced Computer Architecture Lab, 2260 Hayward, Ann Arbor, MI 48109-2121; email: {smita, gviamont, imarkov, jhayes}@eecs.umich.edu.

1. INTRODUCTION AND BACKGROUND

As digital device technology evolves, nondeterministic circuit behavior is becoming more prevalent for several reasons:

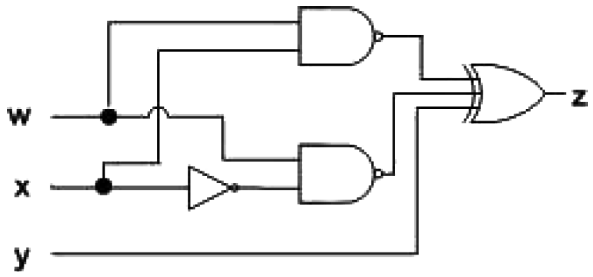
—Single-event upsets induced by external radiation, which can temporarily affect logical functionality.
—Increased process variation and quantization effects in deep-submicron CMOS VLSI circuits (common effects for which many statistical models have been proposed).
—Inherently probabilistic technologies such as quantum computing devices and carbon nanotubes.

The literature on circuit testing has a long history of treating circuits probabilistically. Many papers have dealt with the problem of signal probability estimation [Parker and McCluskey 1975; Ercolani et al. 1989; Savir et al. 1983], which was originally motivated by random pattern testability concerns. The main idea is that the probability of a signal being a 0 or 1 gives some indication of the difficulty in controlling (and therefore testing) the signal. In contrast to signal probability estimation, we deal with complex probabilistic failure modes, error propagation conditions, and their effects on circuit behavior and reliability. Exact circuit reliability evaluation, in general, involves computing not just a single output distribution, but rather the output error probability for each input pattern. In cases where each gate has errors that are input pattern-dependent, even if the input distribution is fixed, simply computing the output distribution does not give the overall circuit error probability. For instance, if only the XOR gate in Figure 1 has an output bit-flip error, then the output distribution is unaffected—but the wrong output is paired with each input. Therefore, we need to compute the error associated with each input vector separately.

Consider the circuit in Figure 1. Given that each gate has error probability $p = 0.1$, the circuit error probability for input combination 000 is 0.244. Input combination 111 has error probability 0.205. The overall error rate of the circuit is the sum of the error probabilities weighted by the input combination probabilities. The probability of error for the circuit in Figure 1, given the uniform input distribution, is 0.225. Note that joint probabilities of input *combinations* rather than individual input probabilities are necessary to capture correlations among inputs.

We reason about circuit reliability and other aspects of probabilistic behavior using the probabilistic transfer matrix (PTM) framework. This framework forms an algebra to represent circuits with probabilistic failure modes, that is, gates exhibiting varying pattern-dependent error probabilities. PTM methods implicitly capture signal correlations caused by reconvergent fanout. These methods are useful in determining the impact of path-based cumulative effects such as glitch attenuation and logic masking on error propagation.

We will first describe PTMs, and then an ADD-based implementation. As is well known, a truth table represents the full range of input/output combinations for a gate, wire, or any logic function. The truth table can be viewed as a 0/1 matrix whose binary row (column) indices correspond to inputs (outputs).



$$(F_2 \otimes F_2 \otimes I)(I \otimes swap \otimes NOT_p \otimes I)(NAND2_p \otimes NAND2_p \otimes I)(XOR3_p)$$

Fig. 1. Sample logic circuit and its symbolic PTM formula.

In such a matrix M for an n-input, m-output logic function f, each entry $M(\mathbf{i}, \mathbf{j})$, with binary indices $\mathbf{i} = i_0 i_1 \ldots i_{n-1}, \mathbf{j} = j_0 j_1 \ldots j_{m-1}$, contains a 1 if the binary inputs $i_0, i_1, \ldots i_{n-1}$ produce the binary output values $j_0, j_1, \ldots j_{m-1}$, and contains a 0 otherwise. This representation is referred to here as an *ideal transfer matrix* (ITM). A *probabilistic transfer matrix* (PTM) is obtained from an ITM, and therefore from a truth table, by allowing each entry to contain any real value in the range $[0, 1]$. Each such value gives the conditional probability that a certain input combination produces a certain output combination, potentially as the result of an error. As we demonstrate, PTMs can represent a wide variety of faults, both probabilistic and deterministic.

The PTMs for logic circuits are constructed from the PTMs of their constituent gates and wires in a systematic way based on their connectivity. The PTMs of gates connected in series are *multiplied*, while the PTMs of gates in parallel are *tensored*, that is, combined using the tensor or Kronecker product. The PTM formula for a circuit provides a concise algebraic representation of the structure and function (both deterministic and probabilistic) of a circuit, akin to a Boolean formula, where $\otimes$ denotes the tensor product. Figure 1 shows a circuit and its PTM formula. Each term in the formula corresponds to a gate or wiring PTM. For instance, $NAND2_p$ is a 2-input NAND gate with output error probability p. The numerical evaluation of a symbolic PTM formula for an entire circuit can produce valuable information about output error probabilities. To aid in such evaluation, we introduce new matrix operations such as *eliminate_variables*, *eliminate_redundant_variables* and *compute_fidelity*.

Limited scalability is often a price that is paid for a general framework that captures complex circuit behavior. Therefore, we develop an implementation of the PTM framework that uses algebraic decision diagrams (ADDs) to compress matrices. We also derive several ADD algorithms that can be used to directly compress and combine the PTMs in order to compute the circuit PTM. Figure 2 gives a PTM for the circuit in Figure 1 along with the corresponding ADD. The PTM represents the situation where each gate of Figure 1 has a probability $p = 0.05$ of an output bit flip. Multiple occurrences of the same value in the PTM of Figure 2 suggests that PTMs are normally compressible. ADDs recognize and eliminate this repetition by retaining only one copy of each distinct PTM entry, although they introduce additional structural nodes. In some cases, ADDs contain exponentially fewer nodes than the number of entries in the explicit

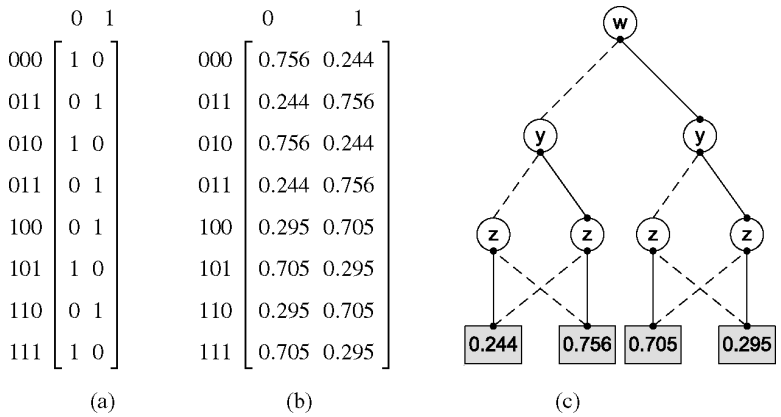


Fig. 2. (a) ITM for the circuit in Figure 1; (b) circuit PTM where each gate experiences error with probability $p = 0.05$; (c) ADD encoding of the PTM.

matrix. In this situation, linear-algebraic transformations can be applied to the ADD exponentially faster than to the matrix.

A major technical challenge is to develop efficient ADD algorithms for PTM operations that operate directly on the compressed forms. As noted above, tensor and matrix multiplication algorithms are needed for PTM-based computation. An ADD algorithm for matrix multiplication is given in Bahar et al. [2003], which involves taking a series of dot products using the APPLY operation. An ADD algorithm for the tensor product is given in Viamontes et al. [2003]. In addition, we develop ADD operations for the new operations mentioned above. These operations are often needed to reconcile dimensions as the algorithms from Bahar et al. [2003] and Viamontes et al. [2003] were originally intended for square matrices.

Recently, a lot of attention has been given to calculating the soft error rate (SER) of a digital circuit [Zhang et al. 2006; Miskov-Zivanov and Marculescu 2006; Zhang and Shanbhag 2004; Dhillon et al. 2005]. Most proposed techniques primarily model masking mechanisms such as logic masking, electrical masking, and latching-window masking in various levels of electrical detail [Shivakumar et al. 2002]. The PTM framework provides a matrix-based mathematical formalism for such work. We illustrate this by showing how to approximate electrical masking explicitly using PTMs. Our goal is technology-independent logical modeling where relevant electrical effects can be approximated if desired.

Various probabilistic approaches have also been applied to symbolic model checking and reachability analysis. These methods are mainly concerned with verifying finite state machines whose transitions are nondeterministic [Hinton and Kwiatkowska 2006]. They do not generally address circuit-specific failure modes although they have been shown to be useful in analyzing Von Neumann's NAND-multiplexing fault-tolerant architecture [Norman et al. 2005].

The main contributions of this work are:

—The matrix-based PTM framework which can represent a wide variety of probabilistic behavior in logic circuits, both symbolically and numerically.

—Rigorous computation and compression methods which involve the encoding of PTMs into ADDs.
—Heuristics that enable the efficient implementation of PTM-based algorithms.
—The application of PTMs to logic-level reliability analysis and gate hardening against soft errors.

The remainder of this article is organized as follows: Section 2 introduces PTMs and their basic operations. Section 3 explains how ADDs are used to compress PTMs and efficiently implement PTM operations. Section 3 also outlines our general PTM evaluation algorithm, and presents empirical results. Section 4 discusses approximations and heuristics that can be used to increase the scalability of PTM-based computation and presents additional empirical results. Section 5 examines several applications. Finally, Section 6 discusses conclusions and future work.

2. PTM THEORY

In this section, we describe the PTM algebra and some key operations to manipulate PTMs and compute reliability. First we discuss the basic operations needed to describe circuits and to compute circuit PTMs from gate PTMs. Next, we define additional operations to extract reliability information, eliminate variables and handle fanout efficiently. Finally we discuss how PTMs capture signal correlation and a wide variety of errors.

2.1 PTM Algebra

Consider a circuit C with n inputs and m outputs. We order the inputs for purposes of PTM representation and label them $in_0, \ldots in_{n-1}$; similarly, the m outputs are labeled $out_0, \ldots out_{m-1}$. The circuit C can be represented by a $2^n \times 2^m$ PTM M. The rows of M are indexed by an n-bit vector whose values range from $\underbrace{000\ldots0}_{n}$ to $\underbrace{111\ldots1}_{n}$. The row indices correspond to truth assignments of the circuit's inputs. Therefore, if $\mathbf{i} = i_0 i_1 \ldots i_n$ is an n-bit vector, then row $M(\mathbf{i})$ gives the output probability distribution for n input values $in_0 = i_0, in_1 = i_1 \ldots\ in_{n-1} = i_{n-1}$. Column indices, similarly, correspond to truth assignments of the circuit's m outputs. If $\mathbf{j}$ is an m-bit vector then entry $M(\mathbf{i}, \mathbf{j})$ is the conditional probability that the outputs have values $out_0 = j_0, out_1 = j_1 \ldots\ out_{m-1} = j_{m-1}$ given input values $in_0 = i_0, in_1 = i_1 \ldots\ in_{n-1} = i_{n-1}$, that is, $P[outputs = \mathbf{j} | inputs = \mathbf{i}]$. Therefore, each entry in M gives the conditional probability that a certain output combination occurs given a certain input combination.

Definition 1. Given a circuit C with n inputs and m outputs, the *probabilistic transfer matrix* for C is a $2^n \times 2^m$ matrix M whose entries are $M(\mathbf{i}, \mathbf{j}) = P[outputs = \mathbf{j} | inputs = \mathbf{i}]$.

Definition 2. A fault-free circuit has a PTM called an *ideal transfer matrix (ITM)*, in which the correct logic value of each output occurs with probability 1.

The PTM for a circuit represents its functional behavior for all input and output combinations. An input vector for an n-input circuit is a row vector with dimensions 1×2^n. Entry $v(\mathbf{i})$ of an input vector v represents the probability that the input values $in_0 = i_0, in_1 = i_1 \ldots in_{n-1} = i_{n-1}$ occur. When an input vector is right-multiplied by the PTM the result is an output vector of size 1×2^m. The output vector gives the resulting output distribution.

Often, PTMs are defined for the gates of a logic circuit. A PTM for the entire circuit can then be derived from the PTMs of its gates and their interconnections. The basic operations needed to compute the circuit PTM from component PTMs are the matrix and tensor products. Consider the circuit C formed by connecting two gates g_1 and g_2 in series; that is, the outputs of g_1 are connected to the inputs of g_2. Suppose these gates have PTMs M_1 and M_2, then the entry $M(\mathbf{i}, \mathbf{j})$ of the resulting PTM M for C represents the probability that g_2 produces output j given g_1 has input i. This probability is computed by summing over all values of intermediate signals (outputs of g_1 which are also inputs of g_2) for input $\mathbf{i}$ of g_1 and output $\mathbf{j}$ of g_2. Therefore each entry $M(\mathbf{i}, \mathbf{j}) = \sum_{all\ \mathbf{l}} M_1(\mathbf{i}, \mathbf{l}) M_2(\mathbf{l}, \mathbf{j})$. This operation corresponds to the matrix product $M_1 M_2$ of the two component PTMs.

Now suppose that circuit C is formed by two parallel gates g_1 and g_2 with PTMs M_1 and M_2. Each entry in the resulting matrix M should represent the joint conditional probability of a pair of input-output values from g_1 and a pair of input-output values from g_2. Each such entry is therefore a product of independent conditional probabilities from M_1 and M_2 respectively. These joint probabilities are given by the tensor product operation.

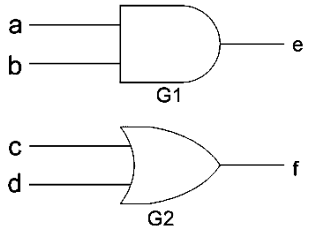
Definition 3. Given two matrices M_1 and M_2 with dimensions $2^k \times 2^l$ and $2^m \times 2^n$ respectively, the *tensor product* $M = M_1 \otimes M_2$ of M_1 and M_2 is a $2^{km} \times 2^{ln}$ matrix whose entries are:

$$\begin{aligned} M(i_0 \ldots i_{k+m-1}, j_0 \ldots j_{l+n-1}) = {} & M_1(i_0 \ldots i_{k-1}, i_0 \ldots j_{l-1}) \\ & \times M_2(i_k \ldots i_{k+m-1}, j_l \ldots j_{l+n-1}) \end{aligned}$$

Figure 3 shows the tensor product of an *AND* ITM with an *OR* ITM. Note that the *OR* ITM appears once for each occurrence of a 1 in the *AND* ITM; this is a basic feature of the tensor product.

Besides the usual logic gates (*AND*, *OR*, *NOT*, etc.), it is useful to define three special gates for circuit PTM computation. These are (i) the n-input identity gate with ITM denoted I_n, (ii) the n-output fanout gate F_n, and (iii) the swap gate *swap*. These wiring PTMs are shown in Figure 4.

An n-input *identity gate* simply outputs its input values with probability 1. It corresponds to a set of independent wires or buffers and has the 2×2 identity matrix as its ITM. Larger identity ITMs can be formed by the tensor product of smaller identity ITMs. For instance, the ITM for a 2-input, 2-output identity gate is $I_2 = I \otimes I$. More generally, $I_{m+n} = I_m \otimes I_n$. An n-output *fanout gate*, F_n, copies an input signal to its n outputs. The ITM of a 2-output fanout gate, shown in Figure 4(b), has entries of the form $F_2(i_0, j_0 j_1) = 1$ where $i_0 = j_0 = j_1$ with all other entries being 0. Therefore, the 5-output fanout ITM F_5 has entries $F_5(0, 00000) = F_5(1, 11111) = 1$, with all other entries 0. Wire



(a)

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

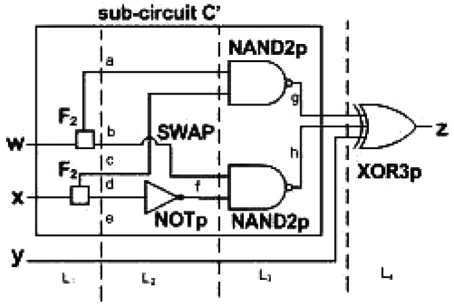
(b)

Fig. 3. Illustration of the tensor product operation: (a) circuit with parallel AND and OR gates; (b) circuit ITM formed by the tensor product of the AND and OR ITMs.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad swap = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) (b) (c)

Fig. 4. Wiring PTMs: (a) identity gate (I) ; (b) 2-output fanout gate (F_2); (c) adjacent swap gate ($swap$).



$$(F_2 \otimes F_2 \otimes I)(I \otimes swap \otimes NOT_p \otimes I)(NAND2_p \otimes NAND2_p \otimes I)(XOR3_p)$$

Fig. 5. Illustration of PTM calculation; vertical lines separate levels of the circuit.

permutations such as crossing wires are represented by *swap gates*. The ITM for an adjacent wire swap (a simple two-wire crossover) is shown in Figure 4(c). Any permutation of wires can be modeled by a series of adjacent swaps.

Example 1. Consider the circuit in Figure 5—this is the same circuit as in Figure 1 with the wiring gates made explicit. The PTMs for the gates with error

probability p are as follows:

$$\begin{bmatrix} p & 1-p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix} \quad \begin{bmatrix} 1-p & p \\ p & 1-p \\ p & 1-p \\ 1-p & p \\ p & 1-p \\ 1-p & p \\ 1-p & p \\ 1-p & p \end{bmatrix} \quad \begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$$

$$NAND2_p \qquad XOR3_p \qquad NOT_p$$

The circuit PTM is expressed symbolically by the formula in Figure 5. Each parenthesized term in the equation corresponds to a level in the circuit. The advantage of evaluating the circuit PTM using such an expression is that the error probabilities for the entire circuit can be extracted from it.

2.2 Additional Operations

In addition to the basic operations of matrix multiplication and tensor product, we introduce the following three operations to increase the scope and efficiency of PTM-based computation:

—*fidelity*: This operation measures the similarity between an ITM and a corresponding PTM. It is used to evaluate the reliability of a circuit.
—*eliminate_variables*: This operation computes the PTM of a subset of inputs or outputs starting from a given PTM. It can also be used to compute the probability of error of individual outputs.
—*eliminate_redundant_variables*: This operation eliminates redundant input variables which result from tensoring matrices of gates that are in different fanout branches of the same signal.

We now formally define and describe these operations in more detail. First, we define the *element-wise* product used in computing *fidelity*.

Definition 4. The element-wise product of two matrices A and B, both of dimension $n \times m$, is denoted $A.*B = M$ and defined by $M(i, j) = A(i, j) \times B(i, j)$.

To obtain the *fidelity*, the element-wise product of the ITM and the PTM is multiplied on the left by the input vector, and the norm of the resulting matrix is computed. In the definition below, $||\mathbf{v}||$ denotes the l_1 norm of vector $\mathbf{v}$.

Definition 5. Given a circuit C with PTM M, ITM J, and input vector $\mathbf{v}$, $fidelity(v, M, J) = ||\mathbf{v}(M .* J)||$.

The fidelity of a circuit is a measure of its *reliability*. Figure 6 illustrates the *fidelity* computation on the circuit from Figure 1. The ITM, shown in Figure 2(a), is denoted J and the PTM, shown in Figure 2(b), is denoted M.

$$v^T = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.25 \\ 0.25 \\ 0 \\ 0 \\ 0.125 \\ 0.125 \end{bmatrix} \quad J.*M = \begin{bmatrix} 0.756 & 0 \\ 0 & 0.756 \\ 0.756 & 0 \\ 0 & .756 \\ 0 & 0.705 \\ 0.705 & 0 \\ 0 & 0.705 \\ 0.705 & 0 \end{bmatrix} \quad v(J.*M) = \begin{bmatrix} 0.371 & 0.371 \end{bmatrix}$$

(a) (b) (c)

Fig. 6. Matrices used to compute *fidelity* for the circuit from Figure 1: (a) input vector; (b) result of element-wise multiplication of ITM and PTM; (c) result of left-multiplication by input vector.

Example 2. Consider the circuit C from Figure 1 with inputs $\{w, x, y\}$ and output $\{z\}$. The circuit PTM is calculated using the PTMs from Example 1 with probability of error $p = .05$ at each gate, on all inputs. Figure 6 shows intermediate matrices needed for this computation. The *fidelity*(v, M, J) is found by first element-wise multiplying J and M, then left-multiplying by an input vector v. The l_1 norm of the resulting matrix is *fidelity*$(v, M, J) = (0.371 + 0.371) = 0.743$. The probability of error is $1 - 0.743 = 0.257$.

The *eliminate_variables* operation is used to compute the "sub-PTM" of a smaller set of input and output variables. We formally define it for 1-variable elimination.

Definition 6. Given a PTM matrix M that represents a circuit C with inputs $in_0 \ldots in_{n-1}$, *eliminate_variables*(M, in_k) is the matrix M' with $n - 1$ input variables $in_0 \ldots in_{k-1} in_{k+1} \ldots in_{k+1} \ldots in_{n-1}$ whose rows are $M'(i_0 \ldots i_{k-1} i_{k+1} \ldots i_{n-2}, \mathbf{j}) = M(i_0 \ldots i_{k-1}\ 0\ i_{k+1} \ldots i_{n-2}, \mathbf{j}) + M(i_0 \ldots i_{k-1}\ 1\ i_{k+1} \ldots i_{n-2}, \mathbf{j})$.

The *eliminate_variables*[1] operation is defined similarly for output variables. The elimination of two variables can be achieved by eliminating each of the variables individually in arbitrary order. Figure 7 demonstrates the elimination of a column variable from a subcircuit C' of the circuit in Figure 5 formed by the logic between inputs w, x and outputs g, h. The PTM for C' with probability of error $p = 0.05$ on all its gates is given by:

$$(F_2 \otimes F_2)(swap \otimes NOT_p)(NAND2_p \otimes NAND2_p).$$

If we eliminate output h, then we can isolate the conditional probability distribution of output g, and vice versa. Output h corresponds to the second variable of the PTM in Figure 7(b). To eliminate this variable, columns 0 and 1 of Figure 7(b) are added and the result is stored in column 1 of Figure 7(c). Columns 2 and 3 of M are also added and the result is stored in column 2. The final PTM gives the probability distribution of output variable g in terms of

[1]The *eliminate_variables* operation is analogous to the existential abstraction of a set of variables x in Boolean a function f [Hachtel and Somenzi 1996], given by the sum of the positive and negative cofactors of f with respect to x: $\exists x\ f = f_x + f_{\overline{x}}$. The *eliminate_variables* operation on PTMs relies on arithmetic addition of matrix entries instead of the Boolean disjunction of cofactors.

$$\begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

(a)

$$\begin{array}{c} \begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 0.0025 & 0.0475 & 0.0475 & 0.9025 \\ 0.0025 & 0.0475 & 0.0475 & 0.9025 \\ 0.04525 & 0.00475 & 0.85975 & 0.09025 \\ 0.09025 & 0.85975 & 0.00475 & 0.04525 \end{bmatrix} \end{array}$$

(b)

$$\begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{bmatrix} 0.0025+0.0475 & 0.0475+0.9025 \\ 0.0025+0.0475 & 0.0475+0.9025 \\ 0.04525+0.00475 & 0.85975+0.09025 \\ 0.09025+0.85975 & 0.00475+0.04525 \end{bmatrix} \end{array}$$

(c)

$$\begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{bmatrix} 0.0025+0.0475 & 0.0475+0.9025 \\ 0.0025+0.0475 & 0.0475+0.9025 \\ 0.04525+0.85975 & 0.00475+0.09025 \\ 0.09025+0.00475 & 0.85975+0.04525 \end{bmatrix} \end{array}$$

(d)

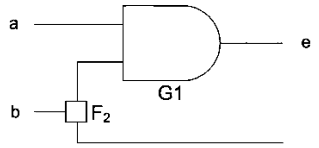
Fig. 7. Example of the *eliminate_variables* operation: (a) ITM of sub-circuit C' from Figure 5; (b) PTM of C'; (c) PTM with output variable h eliminated; (d) PTM with first output variable g eliminated.

the inputs w and x. The same process is undertaken for elimination of the first column variable in the PTM of Figure 7(d).

Often, parallel gates have common inputs due to fanout at an earlier level. An example of this situation appears in level L_3 of Figure 5 due to fanout at level L_1. The fanout gate was introduced to handle such situations; therefore, the level L_1 PTM in Example 1 was composed of fanout PTMs tensored with an identity PTM. However, this method of handling fanout can be computationally inefficient because it requires numerous matrix multiplications. Therefore, we introduce an operation called *eliminate_redundant_variables* to remove redundant signals due to fanout or other causes, in either inputs or outputs. This operation is more efficient than matrix multiplication because it is linear in the size of the matrix, whereas matrix multiplication is cubic.

Definition 7. Given a circuit C with n inputs $in_0, \ldots in_{n-1}$, and PTM M, let in_k and in_l be two inputs that are identified with each other. Then *eliminate_redundant_variables*$(M, in_k, in_l) = M'$ where M' is a matrix with $n-1$ input variables whose rows are $M'(i_1 \ldots i_k \ldots i_{l-1}\ i_{l+1} \ldots i_{n-1}, \mathbf{j}) = M(i_1 \ldots i_k \ldots i_{l-1}\ i_k\ i_{l+1} \ldots i_{n-1}, \mathbf{j})$.

The definition of *eliminate_redundant_variables* can be extended to a set of input variables that are redundant. Figure 8 shows an example of the *eliminate_redundant_variables* operation. PTMs yield correct output probabilities despite reconvergent fanout because the joint probabilities of signals on different fanout branches are computed correctly using the tensor product and the *eliminate_redundant_variables* operations. Suppose two signals on different fanout branches reconverge at the same gate in a subsequent circuit level. Since the joint probability distribution of these two signals is computed correctly, the serial composition of the fanout branches with the subsequent gate is also correct by the properties of matrix multiplication. On the other hand, if the individual signal probabilities are computed separately then these probabilities cannot be recombined into the joint probability without some loss of accuracy.



(a)

$$\begin{array}{c} 000 \\ \mathbf{001} \\ \mathbf{010} \\ 011 \\ 100 \\ \mathbf{101} \\ \mathbf{110} \\ 111 \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)

$$\begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c)

Fig. 8. Signal forwarding using the *eliminate_redundant_variables* operation: (a) circuit with signal b fanning out to two different levels; (b) $NAND \otimes I$, adding b as an input and output; (c) final ITM for circuit computed by removing rows in boldface.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.75 & 0.25 & 0 \\ 0 & 0.25 & 0.75 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 1 & 0 \\ 0.75 & 0.25 \\ 0.25 & 0.75 \\ 0 & 1 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.75^2 & (0.75)(0.25) & (0.25)(0.75) & 0.25^2 \\ 0.25^2 & (0.75)(0.25) & (0.25)(0.75) & 0.75^2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(c)

Fig. 9. Example of output inseparability: (a) PTM for a probabilistic wire-swap; (b) PTM for each individual output after applying *eliminate_variables*; (c) incorrect result from tensoring two copies of the PTM from part (*b*) and applying *eliminate_redundant_variables*.

The *eliminate_redundant_variables* operation can efficiently handle fanout to *different* levels by "signal forwarding" as seen in Figure 8. Signal b is required at a later level in the circuit; therefore, b is added to the ITM as an output variable by tensoring the AND ITM with an identity matrix. However tensoring with the identity ITM adds both an input and output to the level. Hence, the additional input is *redundant* with respect to the second input of the AND gate and removed using *eliminate_redundant_variables*. Note that the removed rows correspond to assigning contradictory values on identical signals.

2.3 Remarks

There are many cases of errors where input and output values cannot be separated, and combinations of these values must be taken into account. For example, the conditional probabilities of the inputs or outputs cannot always be stored separately in different matrices using the *eliminate_variables* operation. While such storage can alleviate the problem of state-space explosion inherent in storing all possible combinations of inputs and outputs, it may not capture correlations within the circuit.

Example 3. Suppose two wires have probability of 0.25 of swapping. The matrix corresponding to this error is given in Figure 9(a). If we try to separate the probability of each output using *eliminate_variables*, the output probabilities both have the PTM of Figure 9(b). If these outputs are tensored (with redundant inputs eliminated) they result in the erroneous combined matrix of Figure 9(c). This demonstrates that these two outputs cannot be correctly separated; their joint conditional distributions are in fact inseparable.

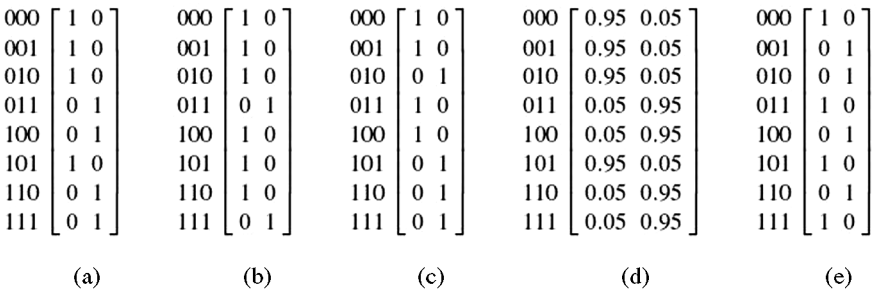


Fig. 10. PTMs for various types of gate errors: (a) a fault-free (ideal) 2-1 MUX gate; (b) first input signal stuck-at 1; (c) first two input signals swapped; (d) probabilistic output bit-flip with $p = 0.05$; (e) wrong gate: MUX replaced by 3-input XOR gate.

Just as some errors cannot be separated, some faults affect multiple gates simultaneously. In this case, the combined PTM cannot be built from individual PTMs and the joint probabilities must be obtained (or the exact correlation determined). This same effect can occur with input vectors, which cannot always be separated into probabilities of individual inputs. An example is given here:

$$\begin{matrix} 00 & 01 & 10 & 11 \\ [\ 0.5 & 0 & 0 & 0.5\ ] \end{matrix}^T$$

PTMs have the advantage that they can represent and manipulate joint probabilities from the inputs to the outputs at every level. When necessary, individual output distributions can be obtained using *eliminate_variables*.

The PTM model can also represent a wide variety of circuit behaviors including stuck-at faults and transient errors. The fact that there are separate probabilities for each input and output, and the fact that they are propagated simultaneously enable this generality. Figure 10 lists a sampling of errors representable by PTMs.

3. COMPUTATION AND COMPRESSION OF CIRCUIT PTMs

The memory needed to store PTMs can be reduced by compressing them and operating on the compressed forms. In this section, we discuss the compression of PTMs using algebraic decision diagrams (ADDs) and develop a procedure for computing circuit PTMs from gate PTMs.

3.1 Compressing Matrices with ADDs

In general, all entries of a $2^n \times 2^m$ PTM can be distinct. However, commonly used PTMs and most ITMs have many identical submatrices. For example, in the ITM of an ideal n-input *AND* gate, all but the last row are identical (see Figure 3). This suggests that circuit matrices can be significantly compressed using decision diagrams.

Bahar et al. [1997] describe the encoding of a matrix using ADDs. Recall that a BDD (binary decision diagram) is a directed acyclic graph representing a Boolean function $f(x_0, x_1, x_2, \ldots x_n)$ with root node x_0. The subtree formed by the outgoing edge labeled 0 represents the cofactor $f_{x_0'}(x_1 \ldots x_n)$ or the *else*

BDD. The subtree formed by the outgoing edge labeled 1 represents the cofactor $f_{x_0}(x_1 \ldots x_n)$, or the *then* BDD. Boolean constants are represented by terminal nodes. ADDs are variants of BDDs where terminal nodes can take on any real value; see Figure 2(c).

The ADD encoding of a matrix M is a rooted directed acyclic graph whose entries depend on the row and column index variables $(r_0, c_0, r_1, c_1 \ldots r_n, c_n)$ of M. The root of the ADD is the node labeled r_0. The subtree formed by the outgoing edge labeled 0 represents the top half of M, i.e., the half corresponding to $r_0 = 0$. The subtree formed by the outgoing edge labeled 1 represents the bottom half of M, which has $r_0 = 1$. Therefore, branches of ADDs correspond to portions of PTMs. As in BDDs, the same path can encode several entries if variables are skipped. The input variables are queried in a predefined order, and this facilitates reductions by using the same node for identical submatrices.

We use the QuIDDPro library [Viamontes et al. 2003] to encode PTMs into ADDs; we also added additional functions to this library for performing operations on PTMs. QuIDDPro includes the CUDD library and uses interleaved row and column variable ordering. This ordering facilitates fast tensor products and matrix multiplications—key operations in the quantum-mechanical simulations for which QuIDDPro was designed. The basic ADD functions used in PTM computations include

—*topvar*(Q) : returns the root node of an ADD Q

—*then*(Q) : returns the 1 branch

—*else*(Q) : returns the 0 branch

—*ITE*(Q, T, E): refers to *if-then-else*. It takes a node Q corresponding to the root, two ADDs T and E corresponding to the *then* and *else* branches, and combines them into a larger ADD.

3.2 Handling Nonsquare Matrices

All matrix algorithms for ADDs that we are aware of assume square matrices, but can represent nonsquare matrices using zero padding [Bahar et al. 1997; Clarke et al. 1996; Viamontes et al. 2003]. A nonsquare matrix has fewer row variables than column variables or vice versa. Recall that ADD variables are ordered, and nodes are levelized by decision variables. Any variable missing from the ADD is interpreted as marking replicated matrix entries. In other words, there is no dependence on the missing variable for the matrix entries, so the matrix entries are identical for both values of the missing variable. Figure 11 illustrates a situation in which missing variables can create ambiguity. Both the matrices in Figure 11 have identical ADDs despite the fact that the matrix on the left has one column variable and the matrix on the right has two column variables. Without zero-padding, these matrices have identical ADDs because the matrix on the right has no dependency on the second column variable. Therefore, the ADD for this matrix has only one column variable as well. To prevent this ambiguity, missing rows or columns can be explicitly padded with zeros.

Figure 12 describes an algorithm for padding matrices with zeros. This algorithm assumes that there are more row variables than column variables;

$$\begin{array}{c} \begin{matrix} 0 & 1 \end{matrix} \\ \begin{bmatrix} p & 1-p \\ p & 1-p \\ p & 1-p \\ 1-p & p \end{bmatrix} \end{array} \qquad \begin{array}{c} \begin{matrix} 00 & 01 & 10 & 11 \end{matrix} \\ \begin{bmatrix} p & 1-p & p & 1-p \\ p & 1-p & p & 1-p \\ p & 1-p & p & 1-p \\ 1-p & p & 1-p & p \end{bmatrix} \end{array}$$

(a) (b)

Fig. 11. PTMs with identical ADDs without zero-padding: (a) Matrix with only one column variable; (b) matrix without dependency on the second column variable.

```
pad_with_zeros(Q, num_row_vars, num_col_vars)
{
    diff = num_row_vars-num_col_vars;
    shift_col_var_labels(Q,diff);
    Result = Q;
    for(i=0;i<diff;i++)
      Result = add_missing_var(Result,i);
    return Result;
}

add_missing_var(Q,i)
{
    new  node  c_i;
    if((topvar(Q)<c_i) & & (then(Q)>c_i))
        T = ITE(c_i,0,else(Q));
        E = ITE(c_i,1,then(Q));
        Result = ITE(topvar(Q),T,E);
    else
        T = add_missing_var(then(Q),i);
        E = add_missing_var(else(Q),i);
        Result = ITE(topvar(Q),T ,E);
    return Result;
}
```

Fig. 12. The *pad_with_zeros* algorithm.

however, it can be easily modified to handle cases with more column variables than row variables. Suppose a PTM with ADD A has 2^{m+1} rows and 2^m columns. The zero-padding of A is done by introducing a new node q with $then(q)$ pointing to the original ADD and $else(q)$ pointing to the zero terminal. In Figure 12 the function *shift_col_var_labels* renames nodes representing column variables by shifting the column variable number up to facilitate the introduction of missing variables into the ADD.

Matrix multiplication and addition are compatible with zero padding [Bahar et al. 1997]; however, the tensor product is not. When the tensor product of two padded matrices A and B is computed, the result has spurious rows of zeros which are carried over from the zero-padding of B. Figure 13 shows an example of an ideal NOT gate tensored with an ideal zero-padded NAND gate and illustrates the incorrect results obtained from tensoring zero-padded matrices.

$$\underset{\text{(a)}}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}} \otimes \underset{\text{(b)}}{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}} = \underset{\text{(c)}}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}$$

Fig. 13. (a) NOT gate ITM; (b) zero-padded NAND gate ITM; (c) their tensor product with incorrect placement of all-zero columns.

Columns 3 and 4 erroneously consist entirely of zeros carried over from the zero-padding of the NAND matrix.

To reconcile tensor products with zero padding we add dummy outputs (copies of input signals) to a gate PTM to equalize the number of inputs and outputs. This makes the PTM square, thereby eliminating spurious zeros. In order to add a dummy output to a gate matrix we can simply "forward" one of its input signals to the output as was done in Figure 8. Dummy outputs can be subsequently removed by eliminating the corresponding column variable. Since *eliminate_variables* removes a variable, it may be necessary to pad the matrix with zeros to retain an equal number of row and column variables. In such cases the zero padding is restored using the algorithm given in Figure 12.

3.3 Computing Circuit PTMs

A general algorithm for PTM computation is presented in Figure 14. First, a gate library is specified by gate PTMs, and a circuit (in BLIF format) is read into a data structure that stores individual gates. Next, gate PTMs are converted to ADDs. The circuit is then topologically sorted from primary outputs to primary inputs and the subsequent computation proceeds by topological level. The ADDs for gates at each level are tensored, zero-tracking is performed, and finally the *eliminate_redundant_variables* operation is applied. The ADD representing each level, called *levelADD* in Figure 14, is then multiplied with *circuitADD*. After all levels are multiplied, the *circuitADD* computation is complete. A detail not shown in Figure 14 is that when a circuit has fanouts to multiple levels, then the gate is placed at the first level at which it is needed, and its output is forwarded to other levels as shown in Figure 8. The intermediate-level ADDs are discarded after they are multiplied with the *circuitADD*. This is important for the scalability of the implementation because *levelADDs* are the tensor products of several gate ADDs and can have large memory complexity.

In place of the fanout gates described in Section 2, we use the *eliminate_redundant_variables* operation from Definition 7, whose ADD implementation is given in Figure 15. By removing each duplicated input signal due to fanout, the number of levels decreases and multiplications are saved. Previously computed partial results of the *eliminate_redundant_variables* operation are stored in a common hash table, which is searched first to avoid traversing down common paths or recomputing existing results. In the pseudo-code,

```
struct gate
{
  PTM[][];
  node *ADD;
  DDmanager M;
  inputs[];
  outputs[];
}

compute_circuit_ADD(Circuit  C, Errors  E)
{
  gate_map[] = read(C);
  levels[] = topological_sort(gate_map);
  gate_map[] = add_errors(gate_map,E);
  for (i=0;i<size(gate_map);i++)
    num_inputs = size(gate_map[i].inputs)
    num_outputs = size(gate_map[i].outputs)
    gate_map[i].ADD = convert_PTM_to_ADD(gate_map[i].PTM);
    gate_map[i].ADD = pad_with_zeros(gate_map[i].ADD,num_inputs,num_outputs);
  for (i=0;i<size(levels);i++)
    level_reordered[] = order_outputs_by_previous_level_inputs(levels[i]);
    for(j=0;j<size(level_reordered);j++)
      current_gate = gate_map[level_reordered[j]];
      levelADD = ADD_tensor(levelADD,current_gate.ADD);
      levelADD = zero_track(levelADD);
    redundant_inputs[] = find_redundant_inputs(level[i]);
    for(k=0;k<size(redundant_inputs);k++)
      levelADD = eliminate_redundant_variables(levelADD,redundant_inputs[k])
    circuitADD = ADD_multiply(circuitADD, levelADD);
    delete(levelADD);
  return circuitADD;
}
```

Fig. 14. Algorithm to compute the ADD representation of a circuit PTM. The *gate* struct stores functional information associated with a gate including the PTM, input names, output names and ADD.

capitalized variables refer to ADDs and lower-case variables refer to nodes. This algorithm for the *eliminate_redundant_variables* operation searches the ADD along all paths for the first of two redundant variables v_1, v_2 with $v_1 < v_2$ in the ADD node ordering. Whenever v_1 is found on a path, we traverse down $then(v_1)$ until v_2 is found. We eliminate the v_2 node and point the preceding node to $then(v_2)$. Next, we traverse down $else(v_1)$ and search for v_2, this time we eliminate v_2 and point the preceding node to $else(v_2)$. This process can be repeated in cases where there are more redundant variables. Both *eliminate_variables* and *eliminate_redundant_variables* are operations that could disturb the equality between row and column variables since they both remove variables. Therefore, it may be necessary to introduce zero-padding by applying *pad_with_zeros* (Figure 12).

After the ADD for the PTM and ITM of a circuit are computed, we can compute the *fidelity* of the circuit to extract reliability information (see Figure 16). This operation is implemented by first taking the element-wise product of the ADD for the ITM with the ADD for the PTM and then performing a depth-first

```
The eliminate_redundant_variables(Q,v1,v2)
{
    if(isconstant(Q))return  Q;
    if(topvar(Q) == v1)
        T = comp_remove_branch(then(Q),v2,1);
        E = comp_remove_branch(else(Q),v2,0);
    else
        if(topvar(then(Q))≤ v1)
            T = comp_remove_branch(then(Q),v2,1);
        else  T = eliminate_redundant_variables(then(Q),v1,v2);
        if(topvar(else(Q))≤ v1)
            E = comp_remove_branch(else(Q),v2,0);
        else  E = eliminate_redundant_variables(else(Q),v1,v2);
    R = ITE(topvar(Q),T,E);
    return R;
}

comp_remove_branch(Q,v2,const)
{
    if(table_lookup(Q,v2,const) != NULL);
      return table_lookup(Q,v2,const);
    if(topvar(Q) == v2)
        if(const==1)return  then(Q);
        else  return  else(Q);
    else  if(topvar(Q) ≤ v2)
        return  Q;
    else
        T = comp_remove_branch(then(Q),v2,const);
        E = comp_remove_branch(else(Q),v2,const);
    R = ITE(topvar(Q),T,E);
    table_insert(R,Q,v2,const);
    return R;
}
```

Fig. 15. The *eliminate_redundant_variables* algorithm.

traversal to sum probabilities of correctness. The traversal of the ADD sums the terminal values while keeping track of skipped nodes. A node which is skipped in an ADD is an indication that the terminal value is repeated a power-of-two times, depending on the skipped variable's ordering. Note the ADD implementations of both *eliminate_redundant_variables* and *fidelity* have complexity linear in the size of the ADDs in their arguments. This is important because, once PTMs are calculated, we cannot resort to decompression since the entire PTM may be large.

Results from calculation of circuit ITMs, circuit PTMs, and reliability (computed using the *fidelity* operation) are listed in Table I. We use the smaller LGSynth 91 and LGSynth 93 benchmarks with independent uniform distributions on all primary inputs. These simulations were conducted on a Linux workstation with a 2GHz Pentium 4 processor. In our experiments CPU time was limited to 24 hours. The run-times and memory requirements are sensitive to the width of a circuit, that is, the largest number of signals at any level, which determines the size of the tensor products and zero-tracking matrices. Empirically, circuits with widths of around 40 signals can be evaluated

```
fidelity(Q1,Q2,v)
{
  Q3 = apply(Q1,Q2,*);
  R = multiply(v,Q3);
  prob = sum_probs(R,1);
  return  prob;
}

sum_probs(Q,mult)
{
    if(table_lookup(Q,mult) != NULL)
        return table_lookup(Q,mult);
    sum = 0;
    mult = mult * 2;
    if(isconstant(Q)) return  value(Q)*mult;
    if(topvar(Q)+1 != topvar(then(Q)))
        multT = mult+2;
    sum=sum + sum_probs(then(Q),multT);
    if(topvar(Q)+1 != topvar(else(Q)))
        multE = mult+2;
    sum = sum + sum_probs(else(Q),multE);
    table_insert(sum,Q,mult);
  return  sum;
}
```

Fig. 16. The *fidelity* algorithm.

efficiently. In these experiments, we calculate entire circuit PTMs which means output combination probabilities are computed for *all* input combinations. If we separate output cones and calculate individual output probabilities, the results would scale much further. However, as discussed before, individual output probabilities cannot always be accurately combined to obtain the overall error probability of a circuit. The number of ADD nodes required for reliability computation on each of the circuits (including all intermediate nodes required in computation) is also listed in Table I. This includes the number of nodes used for the ITM, PTM, and intermediate computations.

Table I gives the overall reliability of the circuits for gate error probabilities of 0.05 and also for one-way gate errors of 0.05. In CMOS gates, an erroneous output value 0 is more likely than an erroneous value 1 because SEUs typically short-circuit power to ground. PTMs can encode this bias easily since error probabilities can be different for different input combinations. Relevant empirical results are given in the "one-way" columns in Table I. Note that circuits with a high output-to-input ratio, such as DECOD.blif, tend to magnify gate errors at fanout stems, and therefore have lower reliability. PTM computation for $p = 0.05$ requires greater memory and longer runtime because less compression is possible. Ideal matrices have large blocks of 0s, which lend themselves to more compression.

4. HEURISTICS FOR INCREASED SCALABILITY

We have presented ADD algorithms for PTM-based computation, but their scalability appears limited due to the possibility of combinatorial explosion in

Table I. Reliability Computation and Performance Statistics on Various Small Benchmarks

Circuit	Characteristics				Prob. Error, $p = 0.05$			No. of	Performance, $p = 0$		Performance, $p = 0.05$	
	Gates	Inputs	Outputs	Width	Two-way	One-way 0	One-way 1	ADD Nodes	Memory(MB)	Time(s)	Memory(MB)	Time(s)
C17	6	5	2	5	0.216	0.145	0.085	2003	1.090	0.002	0.071	0.313
mux	6	21	1	23	0.092	0.036	0.061	13501	26.130	3.109	8.341	2.113
z4ml	8	7	4	20	0.329	0.183	0.183	7009	6.594	1.113	3.030	0.840
x2	12	10	7	23	0.386	0.242	0.188	28486	11.015	2.344	237.926	10.523
parity	15	16	1	23	0.397	0.268	0.268	1957	1.060	0.113	0.337	0.262
pcle	16	19	9	16	0.419	0.122	0.332	546160	28.586	6.160	41.956	4.300
cu	23	14	11	23	0.518	0.172	0.418	105797	13.385	2.176	21.549	3.430
pm1	24	27	17	27	0.624	0.373	0.403	454623	77.661	5.031	21.549	13.340
9symml	44	9	1	37	0.465	0.142	0.062	10466742	4445.700	552.668	5341.000	696.211
xor5	47	5	1	19	0.465	0.261	0.430	46785	46.721	3.539	10556.000	19.582

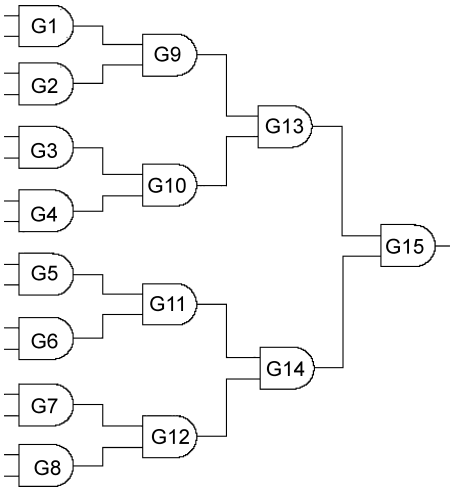


Fig. 17. Tree of AND gates used in Example 4 to illustrate the effect of evaluation ordering on computational efficiency.

PTM sizes. In this section, we develop heuristic approximations that allow the methodology to scale to industry-sized circuits. First, we discuss how dynamic evaluation ordering improves scalability. Next, we demonstrate the use of hierarchy in estimating the reliability of partitioned circuits. Finally, we present reliability calculations using input vector sampling. Note that the role of PTMs remains central to these heuristics because the cumulative effect of various error-prone components is incorporated into the computation via vector-matrix multiplication by PTMs.

4.1 Dynamic Evaluation Ordering

The ADD multiplication algorithm used in QuIDDPro, and adapted from Bahar et al. [2003], has a major impact on the efficiency of PTM computations. The worst-case time and memory complexity of this operation is $O((|A||B|)^2)$ for two ADDs A and B. The PTM evaluation algorithm described in the previous section first tensors gates for each level to form level PTMs, and then multiplies the level PTMs. This creates relatively large multiplication instances. Smaller multiplications can be created by rescheduling the order of evaluation, since delaying the tensor product operation until just before inputs need to be multiplied can result in smaller multiplications.

Example 4. Consider the tree of AND gates in Figure 17. Suppose we wish to compute its circuit PTM. The algorithm of Figure 14 requires topologically sorting the circuit, calculating the PTM for each level and multiplying the levels in order. The levels are $L_4 = \{G15\}, L_3 = \{G14, G13\}, L_2 = \{G12, G11, G10, G9\}, L_1 = \{G8, G7, G6, G5, G4, G3, G2, G1\}$. The level PTMs have dimensions $2^2 \times 2$, $2^4 \times 2^2$, $2^8 \times 2^4$, and $2^{16} \times 2^8$ respectively. First we compute $L_3 \times L_4$ which is of dimension $2^4 \times 2$. Next, L_2 is multiplied by $L_3 \times L_4$, yielding a matrix of size $2^8 \times 2$ and so on. The following matrix products are performed: $(2^2 \times 2, 2^4 \times 2^2)$, $(2^4 \times 2, 2^8 \times 2^4)$, $(2^8 \times 2, 2^{16} \times 2^8)$. In the worst case,

Table II. Comparison of Runtimes and Memory Usage for Levelized Ordering and Ordering Computed by Dynamic Programming

	Improved Ordering		Levelized Ordering	
Circuit	Time(s)	Memory(MB)	Time(s)	Memory(MB)
C17	0.212	0.000	1.090	0.004
mux	18.052	2.051	26.314	3.109
z4ml	3.849	1.004	6.594	1.113
x2	11.015	2.344	193.115	12.078
parity	1.060	0.113	1.07	0.133
pcle	28.810	3.309	98.586	6.160
decod	5.132	1.020	30.147	24.969
cu	23.700	2.215	13.385	2.176
pm1	72.384	3.734	77.661	5.031
cc	57.400	4.839	1434.370	155.660
9symml	89.145	6.668	4445.670	552.668
xor5	3.589	0.227	46.721	3.539
b9	9259.680	165.617	23164.900	295.984
c8	35559.500	930.023	mem-out	mem-out

the ADD sizes are close to matrix sizes (in general, they are smaller as ADDs provide compression), so the total cost of matrix multiplication is $2^{50}+2^{34}+2^{18}$. On the other hand, separating the gates (not tensoring) for as long as possible starting from the primary inputs yields the following multiplication instances: $4(2^4 \times 2^2, 2^2 \times 2)$, $2(2^4 \times 2, 2^2 \times 2)$, $(2^8 \times 2, 2^2 \times 2)$. Here, the total multiplication cost is only $2^{20}+2^{27}+2^{42}$. Therefore, scheduling matrix multiplication carefully leads to a more efficient PTM computation algorithm.

If the output of a source gate is connected to more than one sink gate, there are two possibilities: the first is to tensor these gate PTMs and eliminate the redundant variables; the second possibility is to process these gates and keep logic cones separate until they are naturally tensored at some future level. We choose the latter approach, which exchanges multiplications for tensor products. This is advantageous as the tensor product has lower complexity than multiplication. Determining the optimal order to multiply levels is similar to solving the matrix chain multiplication problem [Cormen et al. 2001], which can be solved by a dynamic programming algorithm in $O(n^3)$ time. Our application can use the same algorithm if the cost of multiplying two matrices is estimated based on their dimensions without taking ADD compression into account.

The results of applying the improved ordering for multiplication of levels are given in Table II. Values in this table were produced on a Pentium 4 Xeon processor running at 2GHz. In general this ordering method uses less memory with a modest increase in runtimes. The runtime increase is partially due to the overhead of the dynamic programming. However since memory was the main bottleneck previously, this ordering stops the PTM evaluation program from thrashing on some larger benchmarks.

4.2 Hierarchical Reliability Estimation

In this section, we extend PTM analysis hierarchically to estimate the reliability of larger circuits partitioned into subcircuits. First the ITMs and PTMs of all subcircuits are calculated. Then, in topological order, we calculate the

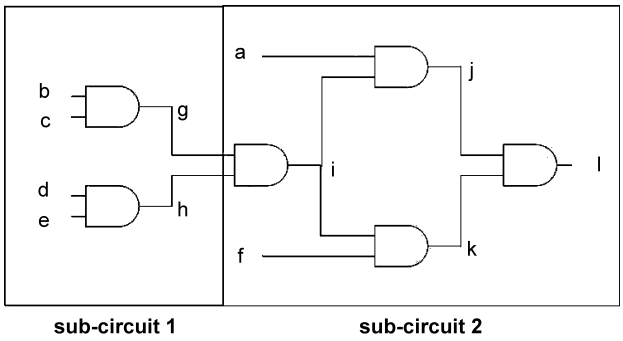


Fig. 18. Circuit used in Example 5 to illustrate hierarchical reliability estimation.

bit fidelities and output distributions of each of the subcircuit outputs. Since evaluation proceeds in topological order, input fidelities are already calculated for previously processed subcircuits.

Consider the circuit in Figure 18. We process subcircuit 1 first using the primary input distributions of b, c, d, and e. We multiply the joint input distribution of b, c, d, and e by the PTM for subcircuit 1 to obtain the bit-fidelities and distributions on g and h. These are, in turn, used to form the input distribution for subcircuit 2, along with a and f. After subcircuit 2 is processed we will obtain the bit fidelity of the primary output i.

In order to formally define bit fidelity we introduce the *abstract* operation for notational convenience.

Definition 8. For a PTM M and an output variable o_k, $M' = abstract(M, k)$ is the matrix which results from the elimination of all variables *except* o_k from M. Therefore $M' = eliminate\_variables(M, 0, 1, 2 \ldots k-1, k+1 \ldots m)$

Definition 9. The *bit fidelity* of output o_k of circuit C, with ITM J, PTM M, and input distribution v is the probability of error of the kth output bit, given by $bit\_fidelity(k, v, J, M) = fidelity(v_k, J_k, M_k)$, where $J_k = abstract(J, k)$, $M_k = abstract(M, k)$ and $v_k = abstract(v, k)$

Suppose the input bit fidelities for a particular subcircuit are p_1, p_2, $p_3 \ldots p_n$. Then, in order to account for input error, the subcircuit PTM is multiplied by $I_{p_1} \otimes I_{p_2} \ldots I_{p_n}$ where I_p has the form $\begin{bmatrix} p & 1-p \\ 1-p & p \end{bmatrix}$.
The probability distribution of each signal is also calculated by multiplying the input distribution of each subcircuit by its ITM and then abstracting each of the output probabilities. The algorithm details are given in Figure 19, where *SubCircArray* is the topologically sorted array of subcircuits, *PIs* is the list of primary inputs, *POs* is the list of primary outputs, *Distro* stores the separated probability distribution of intermediate variables, and the *Bfid* array contains bit fidelities of previously processed signals. At each iteration *Bfid* is updated with output fidelities of the current subcircuit. At the termination of the algorithm *Bfid* will contain the bit fidelities of the primary outputs.

This algorithm has several interesting features. First, it only calculates PTMs of subcircuits and thus avoids the state space explosion associated with

```
estimate_bit_fidelity(input v, circuit C)
  SubCircArray[] = topological_sort(C);
  PIs[]=get_primary_inputs(C);
  POs[]=get_primary_outputs(C);
  for(i=0;i<size(PIs);i++)
    Bfid[PIs[i]]=0;
    Distro[PIs[i]]=abstract(v,i);
  for(i=0; i<size(SubCircArray);i++)
    SPOs[]=get_primary_outputs(SubCircArray[i]);
    SPIs[]=get_primary_inputs(SubCircArray[i]);
    J=itmcalc(SubCircArray[i]);
    M=ptmcalc(SubCircArray[i]);
    vin_i = Distro[SPIs[1]] ⊗ Distro[SPIs[2]] ⊗...;
    vout_i = vin_i * J;
    for(j=0;j<size(SPOs);j++)
     Distro[SPOs[j]]=abstract(vout_i,j);
     M'= (I(Bfid[SPIs[1]])⊗ I(Bfid[SPIs[2]]) ...) M;
     Bfid[SPOs[j]]=bit_fidelity(j,Distro[SPOs[j],J,M');
  return Bfid;
```

Fig. 19. The *Bit_fidelity* estimation algorithm.

directly computing the entire circuit PTM. For instance, if a circuit with n inputs and m outputs is partitioned into two subcircuits each with $n/2$ inputs and $m/2$ outputs, the PTMs of the two subcircuits together are of size $2(2^{(n+m)/2})$, which is significantly smaller than the circuit PTM, which has size 2^{n+m}. Second, the heuristic approximates joint probability distributions using marginal probability distributions and averages local error probabilities at each subcircuit. Any loss of accuracy is a result of the *abstract* operation and the averaging effect that occurs in *bit fidelity* calculations. Therefore, the estimation technique will be very accurate in cases where there is no reconvergent fanout between the subcircuits. In fact, the heuristic is exact when each output bit has the same error on all input combinations because in such cases averaging does not result in a loss of information. In other cases, the accuracy will depend on the amount of correlation between signals, and the variation in signal errors.

Example 5. We apply the algorithm of Figure 19 to the circuit in Figure 18. Assume that each of the AND gates in Figure 18 has the following PTM and ITM:

$$AND2_{0.1} = \begin{bmatrix} 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.9000 & 0.1000 \\ 0.1000 & 0.9000 \end{bmatrix} \qquad AND2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Suppose that primary inputs are uniformly distributed and have no errors. Initialize $Bfid[a] = Bfid[b] = Bfid[c] = Bfid[d] = BFid[e] = Bfid[f] = 1$ and $Distro[a] = Distro[b] = Distro[c] = Distro[e] = Distro[f] = [0.5\ \ 0.5]$. The input vector for subcircuit 1 is given by:

$$vin_1 = [0.0625\ \ 0.0625\ \ 0.0625\ \ 0.0625 \ldots 0.0625].$$

The PTM and ITM for subcircuit 1 are calculated as follows:

$$ITM1 = AND2 \otimes AND2$$

$$PTM1 = AND2_{0.1} \otimes AND2_{0.1}$$

$$ITM1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad PTM1 = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix}.$$

The fidelity and probability distribution for each output of subcircuit 1 are calculated as follows:

$$vout_1 = vin_1 * ITM1 = [0.5625 \;\; 0.1875 \;\; 0.1875 \;\; 0.0625]$$

$$Distro[g] = abstract(vin_1, g) = [0.75 \;\; 0.25]$$

$$Distro[h] = abstract(vin_1, h) = [0.75 \;\; 0.25]$$

$$PTM1' = (I(1) \otimes I(1) \otimes I(1) \otimes I) * PTM1 = PTM1$$

$$Bfid[g] = bit\_fidelity(g, Distro[g], PTM1', ITM1) = 0.9$$

$$Bfid[h] = 0.9.$$

Similarly for subcircuit 2:

$$ITM2 = (I \otimes AND2 \otimes I)(I \otimes F_2 \otimes I)(AND2 \otimes AND2)(AND2)$$

$$PTM2 = (I \otimes AND2_{0.1} \otimes I)(I \otimes F_2 \otimes I)(AND2_{0.1} \otimes AND2_{0.1})(AND2_{0.1})$$

$$PTM2' = (I \otimes I_{0.9} \otimes I_{0.9} \otimes I)(PTM2)$$

$$vin_2 = [0.5 \;\; 0.5] \otimes [0.75 \;\; 0.25] \otimes [0.75 \;\; 0.25] \otimes [0.5 \;\; 0.5]$$

$$ITM2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad PTM2 = \begin{bmatrix} 0.8920 & 0.1080 \\ 0.8856 & 0.1144 \\ 0.8920 & 0.1080 \\ 0.8856 & 0.1144 \\ 0.8920 & 0.1080 \\ 0.8856 & 0.1144 \\ 0.8920 & 0.108 \\ 0.8344 & 0.1656 \\ 0.8856 & 0.1144 \\ 0.8280 & 0.1720 \\ 0.8856 & 0.1144 \\ 0.8280 & 0.1720 \\ 0.8856 & 0.1144 \\ 0.8280 & 0.1720 \\ 0.8344 & 0.1656 \\ 0.3160 & 0.6840 \end{bmatrix} \quad PTM2' = \begin{bmatrix} 0.8920 & 0.1080 \\ 0.8851 & 0.1149 \\ 0.8920 & 0.1080 \\ 0.8810 & 0.1190 \\ 0.8920 & 0.1080 \\ 0.8810 & 0.1190 \\ 0.8920 & 0.1080 \\ 0.8441 & 0.1559 \\ 0.8851 & 0.1149 \\ 0.8229 & 0.1771 \\ 0.8810 & 0.1190 \\ 0.7819 & 0.2181 \\ 0.8810 & 0.1190 \\ 0.7819 & 0.2181 \\ 0.8441 & 0.1559 \\ 0.4133 & 0.5867 \end{bmatrix}$$

$$vout_2 = [0.9922 \quad 0.0078]$$

$$Distro[l\,] = [0.9922 \quad 0.0078]$$

$$BFid[l\,] = bit\_fidelity(l\,, Distro[l\,], PTM2', ITM2) = 0.869.$$

Alternatively, calculating the fidelity using the circuit PTM gives *fidelity* = 0.862. This has an error of only 0.003 for gate errors in the range 0.1.

The *fidelity* of the entire circuit (rather than just its output bits) can be further estimated by calculating the probability that *any* of the output bits have any error using the binomial probability distribution. This once again assumes that output signals are independent.

4.3 Input Vector Sampling

Reliability estimation requires computing the error associated with each input combination. One way to approximate this is to evaluate a sampling of input vectors. Evaluating the output distribution under specific input vectors involves a series of vector-matrix multiplications where vectors representing signals are multiplied by gate PTMs. The advantage of using the gate PTMs in this process is that the cumulative effects of complex error modes are processed as signals travel through the logic circuit.

The complexity of this method is due to the signal correlations caused by reconvergent fanout. To improve accuracy, we store outputs of the same multiple-output gate or the same fanout branch as joint probability distributions. Then, when a gate only uses one of a set of k correlated signals as an input, the

Table III. Reliability Estimation Using Input Vector Sampling on ISCAS85 Circuits

	Characteristics			Performance, $p = 0.1$		Reliability
Circuit	Gates	Inputs	Outputs	Time(s)	Memory (MB)	One-Way
b9	117	41	21	15.7	1.74	0.257
C432	160	36	36	161.5	6.4	0.369
C499	202	41	32	18.7	2.4	0.221
C880	383	60	26	30.0	5.7	0.320
C1908	880	33	25	220.7	11.5	0.373
C2670	1193	233	140	72.7	17.55	0.128
C3540	1669	59	22	330.94	21.2	0.436
C5315	2307	178	123	233.62	31.457	0.303

gate PTM is enlarged by tensoring with an I_{k-1} to process the additional correlated inputs. This way correlated signal probabilities are never separated. This method has complexity that is linear in the circuit size. The associated constant is related to the maximum gate size multiplied by the maximum number of signals stored jointly. In other words, for a circuit with N gates where the largest gate has n inputs, if we store a maximum of K correlated signals jointly, the complexity of input vector sampling is $O(2^{nK}N)$.

Note that this is different from the hierarchical reliability estimation algorithm of Section 4.2. In the extreme case of that algorithm, each gate is in its own partition and the error probability for the particular input distribution and output distribution is computed for each gate. In essence, the error probability for the input distribution is propagated through to the output and no sampling is required. However, the loss of accuracy is a result of the marginalization that occurs in signals between partitions since inputs to partitions are treated as pseudo-primary. In contrast, here we sample a selection of input vectors. We maintain accuracy by storing correlated signals jointly as much as possible. Benchmark results for this method are given in Table III. These results were calculated while storing a maximum of 10 signals jointly. The input vectors in Table III were chosen uniformly at random. Results are shown for average run times and probabilities of error for 100 samples.

5. APPLICATIONS

In this section, we obtain various kinds of information related to circuit reliability using PTMs. In Section 5.1, we analyze circuit reliability as a function of gate reliability. Using data points for various gate error values, we derive low-degree polynomial approximations for the error transfer functions of standard benchmark circuits. Such functions can be used to derive upper bounds for tolerable levels of gate error. In Section 5.2 we identify the gates in a circuit that are most susceptible to error. Finally, Section 5.3 discusses SEU modeling where electrical attenuation effects are incorporated into gate PTMs.

5.1 Circuit Error Transfer Function

In [Krishnaswamy et al. 2005] we applied PTMs to von Neumann's NAND-multiplexer circuit [von Neumann 1956] in order to calculate how changes in the number of signal replications, the error probability, and the number of levels

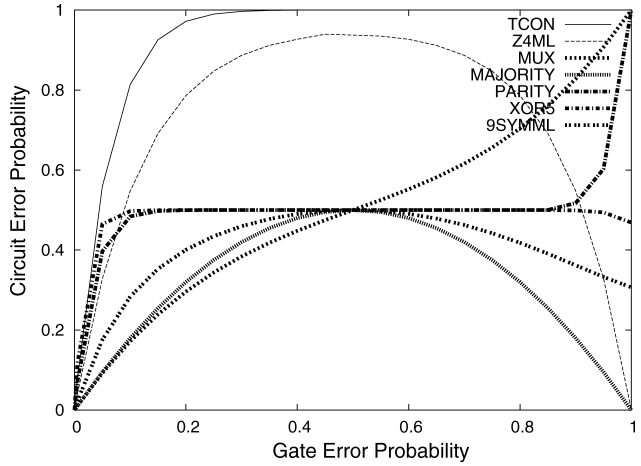


Fig. 20. Circuit error probability under various gate error probabilities.

Table IV. Polynomial Approximations of Circuit Error Transfer Curves and Residual Errors
(The fitted polynomials are of the form $e(x) \approx a_0 + a_1x + a_2x^2 + a_3x^3 \ldots$.)

Circuit	Error	Polynomial Coefficients						
		a_0	a_1	a_2	a_3	a_4	a_5	a_6
majority	2.5 E-7	0.2080	0.1589	0	0	0	0	0
mux	6.6 E-6	0.0019	1.9608	−2.8934	1.9278	0	0	0
parity	0.0040	0.0452	5.4892	−21.4938	31.9141	−4.2115	−30.3778	19.5795
tcon	0.0019	0.0152	6.2227	−13.5288	7.1523	9.2174	−9.0851	0
9symml	0.0010	0.0250	2.4599	−3.7485	1.5843	0	0	0
xor5	0.0043	0.0716	5.9433	−26.4666	51.1168	−44.6143	14.4246	0

affects the output reliability. In this section, we use PTMs to extend this type of analysis to general combinational circuits. The irregularity of general combinational circuits makes their reliability analysis much more computationally complex.

Definition 10. The *error transfer function* $e(x)$ of a circuit C, on $0 \leq x \leq 1$, is the *fidelity* of C with output error probability x on all gates.

Figure 20 illustrates the error transfer functions for several standard benchmark circuit, determined by introducing varying amounts of error into gates and then calculating the circuit fidelity according to Definition 5. Generally, such error transfer curves can be described by polynomials. If two gates have error p then their composition (serial, parallel, or a combination of both) has terms that are linear combinations of p^2 and p. The overall probability of error is $O(p^2)$. If a circuit has n gates, each with error p, then its fidelity is a polynomial in p of degree n. Realistically, only gate error values under 0.5 are useful since the gate can simply be viewed as its negated version for higher error values. However, Figure 20 has probabilities of gate error up to 1 to make the polynomial nature of the curves evident.

Table IV gives low-degree polynomials that estimate error transfer functions with high accuracy. Such functional approximations are useful in determining

Table V. The Average Probability that Errors Occuring with $p = 0.1$ Propagate to the Output in Various Circuits

Circuit	Error
parity	0.010
pcle	0.067
z4ml	0.010
xor5	0.056
tcon	0.010
C17	0.082

upper bounds on the gate error probability necessary to achieve acceptable levels of circuit error. For instance, it has been shown that replication techniques such as TMR or NAND-multiplexing only decrease circuit error if the gate error is strictly less than 0.5 [Pippenger 1998]. However, Figure 20 suggests that for most circuits, replicating the entire circuit at gate errors of 0.20 or more will only increase circuit error.

5.2 Error Susceptibility

Next, we examine the ability of a circuit to mask internal errors. We show that, in general, circuit errors can be significantly overestimated when logic masking is not considered. Since circuits mask errors at different locations with different probabilities, we also define a measure of the susceptibility of a signal to error.

Table V shows the effect of logic masking averaged over all gates in some representative circuits. For this experiment, we introduce an error of 0.1 at all inputs of a particular gate and then calculate the resulting circuit error probability. We then repeat this for all the gates in the circuit and present the average output error probability in Table V. On average, not taking logical masking into consideration appears to overestimate circuit error by a factor of 10.

As is well known, gate location and size influence error propagation and logic masking. For instance, errors at a primary output line have no chance of being masked. Identifying and replicating gates that are highly susceptible to soft errors can reduce the amount of internal redundancy needed for reliable operation [Mohanram and Touba 2003]. PTMs can provide an *exact* measure of the susceptibility of the circuit to errors at specific gates. This is in contrast to other methods such as that in [Mohanram and Touba 2003], where test vector sampling is used for this purpose.

Definition 11. Given a circuit C with ITM J, an internal gate g, input vector v, and a PTM M computed by adding an error p to gate g only (with all other gates being error-free), the *susceptibility* of C at gate g is:

$$susceptibility(C, g) = fidelity(v, M, J)/p.$$

The susceptibility of a circuit to a gate error can be computed by introducing an error probability into the appropriate gate PTM and then evaluating the fidelity of the corresponding circuit. The gates with highest susceptibility can be regarded as those that affect the overall error the most. Since PTM calculations simultaneously include all input vectors, sampling is unnecessary.

```
compute_susceptibility(gate g, circuit C)
{
    perr = 1;
    compute_intermediate_PTMs(C);
    currlevelPTM = compute_level_PTM(C,level(g),perr);
    errorPTM = leftPTM[level(g)-1]*currlevelPTM*rightPTM[level(g)+1];
    idealPTM = leftPTM[numlevels(C)-1];
    return gate_fidelity(errorPTM,idealPTM);
}

compute_intermediate_PTMs(circuit C)
{
    perr = 0;
    for(i=0;i<numlevels(C);i++)
      levelPTM[i]=compute_level_PTM(C,i,perr);
    for(i=0;i<numlevels(C);i++)
      leftPTM[i] = leftPTM[i-1]*levelPTM[i]
      rightPTM[i] = levelPTM[n-i]*rightPTM[i-1]
}
```

Fig. 21. The *compute_susceptibility* algorithm.

Table VI. Improvement in *fidelity* After Increasing Robustness of the Top 3 and 5 Most Susceptible Gates

Circuit	Original *fidelity*	Top 3 gates	% Improvement	Top 5 Gates	% Improvement
C17	0.864	0.959	11.0 %	0.980	13.4 %
mux	0.907	0.974	7.39 %	0.985	8.60 %
parity	0.603	0.637	5.64 %	0.666	10.4 %
xor5	0.047	0.068	46.2 %	0.070	50.5 %
pm1	0.375	0.429	14.4 %	0.469	25.1 %

The simplest way to compute susceptibility of a circuit to an error at gate g is to introduce an error on g and leave all other gates in the fault-free ideal state. However, for a circuit with n gates this method requires n circuit PTM computations. In order to reduce time complexity, intermediate results can be cached in such a way that only two PTM evaluations are necessary for computing the most susceptible gates in the circuit. In this method the PTM evaluation is done twice, once in increasing level order and once in decreasing level order. The intermediate results are stored. When a gate g at level i is evaluated for susceptibility, the *level*(i)-PTM is recomputed and multiplied by the precomputed level-$(0, i-1)$ and level-$(i+1, k)$ PTMs. The computation of a level i PTM can be similarly simplified by storing intermediate tensors of gate subsets. Therefore each gate susceptibility computation will require two tensor products and two matrix multiplications instead of an entire PTM evaluation. This algorithm is shown in Figure 21.

Table VI illustrates gate susceptibility calculations for several small circuits. The most susceptible gates are identified using the algorithm in Figure 21 and subsequently "hardened" by a factor of 10. All other gates retain error probability 0.05. Hardening can be implemented by sizing up transistors so that they can only be affected by higher-energy particles. However, such transistors occupy larger area and require more power. For many circuits, increasing

the robustness of just a few gates can improve circuit reliability significantly [Mohanram and Touba 2003].

5.3 Modeling Electrical Glitch Attenuation

We now demonstrate how the PTM model can be used to determine error rates under SEUs. Recall that SEUs are mitigated by three types of error masking: logical, electrical and timing masking [Shivakumar et al. 2002]. One of the main challenges in estimating the error rate is determining the logical dependencies between the three types of masking. In Section 5.2 we demonstrated that PTMs can evaluate the impact of logic masking on errors throughout the circuit. We now incorporate the effects of electrical attenuation into PTMs as well. We first discuss a specific model of electrical attenuation presented in Omana et al. [2003] and incorporate it directly into PTMs. This model has been validated using SPICE and was shown to be over 90% accurate. The authors of Omana et al. [2003] classify erroneous glitches into three types based on their duration D relative to the gate propagation delay T_p.

—Type 1: If $D > 2T_p$, the glitch passes through un-attenuated because there is sufficient energy to propagate it. The output amplitude in this case is $A' = V_{dd}$.

—Type 2: If $2T_p > D > T_p$, the glitch propagates with amplitude A diminished by attenuation to $A' = V_{dd}/(VT_1 - VT_2) * A(V_{dd}/2 - VT_1)$. Note that if $A' < V_s$, where V_2 is the threshold voltage, then this glitch no longer has the amplitude to cause a logical error. Hence, some glitches of Type 2 are also electrically masked.

—Type 3: If $T_p > D$, the glitch will not propagate at all. Hence in this case $A' = 0$.

Let the probability that a glitch is of type $i = 1, 2, 3$ be P_i. As in Omana et al. [2003], we assume a uniform distribution of glitch duration D in the range $[0, 2T_p]$, but other distributions can be handled by integration. Also let $P(A)$ be the probability distribution of the glitch amplitude A. The probability that a glitch becomes attenuated by a gate is given by:

$$P_{att} = P_3 + P_2 * P(A' < V_s) \tag{1}$$

Since this model is discrete and probabilistic, it can be abstracted into a logical form and incorporated into the gate PTM model using an additional input bit representing the glitch type. In the resulting PTM, the first bit indicates the logic value, that is, whether the amplitude A of the glitch is greater than the gate threshold voltage. The second bit indicates the duration (long or short) of the glitch. Since glitches of type 3 do not propagate, we only need to differentiate between the first two types. Therefore, glitches of type 1 are represented by 11, and glitches of type 2 are represented by 10. Glitches with amplitude smaller than the logic threshold value can be represented by 01 regardless of duration. According to this model, 10 signals are transformed into 00 signals with a certain probability. All other signals retain their original output value given by the logic function of the gate. Figure 22 illustrates the attenuation PTM for the identity gate and the two-input AND gate. Calculating the circuit

$$
\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array}
\begin{array}{c}
\begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\
\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]
\end{array}
$$

(a)

$$
\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array}
\begin{array}{c}
\begin{array}{cccc} 0001 & 10 & 11 \end{array} \\
\left[\begin{array}{ccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & P_{att} & 1-P_{att} \\ 0 & 0 & 0 & 1 \end{array}\right]
\end{array}
$$

(b)

$$
\begin{array}{c} \\ 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0100 \\ 0101 \\ 0110 \\ 0111 \\ 1000 \\ 1001 \\ 1010 \\ 1011 \\ 1100 \\ 1101 \\ 1110 \\ 1111 \end{array}
\begin{array}{c}
\begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\
\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]
\end{array}
$$

(c)

$$
\begin{array}{c} \\ 0000 \\ 0001 \\ 0010 \\ 0011 \\ 0100 \\ 0101 \\ 0110 \\ 0111 \\ 1000 \\ 1001 \\ 1010 \\ 1011 \\ 1100 \\ 1101 \\ 1110 \\ 1111 \end{array}
\begin{array}{c}
\begin{array}{cccc} 00 & 01 & 10 & 11 \end{array} \\
\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & P_{att}^2 & (1-P_{att}^2) & 0 \\ 0 & P_{att} & (1-P_{att}) & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & P_{att} & (1-P_{att}) & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]
\end{array}
$$

(d)

Fig. 22. PTMs modeling buffers and AND gates with glitch-attenuating properties: (a) $I_{att\text{-}ideal}$, (b) I_{att}, (c) $AND_{att\text{-}ideal}$, (d) AND_{att}.

PTM with the attenuation PTMs from Figure 22 automatically tracks signals on sensitized paths only. In order to describe the probabilities of SEU strikes at each circuit node, we can define a *glitch creation PTM* which gives the probability distribution of glitch generation at a particular node; see Figure 23.

Observe that if different gates have vastly different propagation delays, then the relative probabilities of glitches of each type will be different. This effect may need to be taken into account by remapping signal probabilities in neighboring gates based on their relative propagation delays. Such a remapping can be done using a modified identity matrix at each fanout stem, which does not significantly change the complexity of PTM evaluation.

Note also that PTMs can readily handle multiple glitches and reconvergence. For instance, the row with index 1010 has two inputs with logic-1 values of type 2 which can represent glitches arriving at both outputs. In Figure 22, we assumed that an error propagates if one or both glitches propagate. More complex glitch models, such as the one in Mohanram and Touba [2003] can also be used to derive P_{att} if desired.

Example 6. For the circuit in Figure 24, suppose an *SEU* strike produces a glitch at input b. By inspection, we see that this glitch will only logically propagate for the primary input combination 101. In this case, the glitch passes through both AND gates. Therefore, the probability that the glitch causes an

error, averaged over all inputs, is:

$$(1/8) * P_{strike}(P_1 + P_2 * (1 - P_{att})^2) = .000052083$$

In other words, the glitch propagates if the input sensitizes the appropriate path and the glitch propagates to d and then e. If we let $P_1 = P_2 = P_3 = 0.333$, $P_{strike} = 0.001$, $P_{att} = 0.5$, and AND_{att} is as shown in Figure 22, then the circuit PTM is given by:

$$(I_2 \otimes I_{strike} \otimes I_2)(AND_{att} \otimes I_2)(AND_{att})$$

The corresponding PTM and fidelity are given in the following.

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
\vdots & & & \\
1 & 0 & 0 & 0 \\
0.9996 & 0.0001 & 0.0003 & 0 \\
0.9996 & 0.0002 & 0.0002 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.2500 & 0.1875 & 0.5625 & 0 \\
0.2500 & 0.3750 & 0.3750 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.5000 & 0.1250 & 0.3750 & 0 \\
0.5000 & 0.2500 & 0.2500 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0.9995 & 0.0002 & 0.0003 & 0 \\
0.9995 & 0.0001 & 0.0001 & 0.0003 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0.1250 & 0.3750 & 0 \\
0 & 0.2500 & 0.2500 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0.5000 & 0.5000 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}$$

$$fidelity = .99994791$$

$$Perror = 1 - fidelity = .000052083$$

$$I_{strike} = \begin{bmatrix} 1 - P_{strike} & P_1 P_{strike} & P_2 P_{strike} & P_3 P_{strike} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 23. Glitch creation PTM; P_i denotes the probability of a glitch type i, and P_{strike} denotes the probability of an SEU.

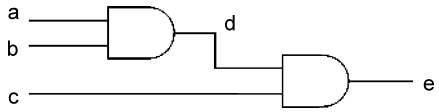


Fig. 24. Circuit used in example 6 to illustrate the incorporation of electrical masking into PTMs.

This example demonstrates that a PTM-based approach can be used to discover the logical paths through which a glitch propagates and the probability of its propagation. For this purpose, gate attenuation properties have to be converted into probabilities of error, which can be done straightforwardly as in Equation 1.

6. CONCLUSIONS

In this work we used PTMs to develop a powerful new methodology for representing probabilistic behavior in logic circuits. PTMs provide a rigorous algebraic representation of a circuit's structural and functional information. An exceptionally wide variety of errors, both deterministic and probabilistic, can be represented by PTMs. They subsume deterministic fault models like the stuck-at model and can be used to model glitch propagation at the logic level. PTMs also allow for the computation of useful functions like circuit fidelity, in a systematic and computationally efficient manner.

We implemented PTMs using ADD-based compression and developed algorithms which operate directly on the compressed forms. PTM computation has a wide variety of applications in circuit reliability. It can be used to determine the error transfer behavior and error susceptibility of combinational circuits. In addition, PTM-based reliability evaluation can be extended hierarchically to evaluate arbitrarily large circuits with a small loss in accuracy. PTMs can also incorporate electrical phenomena such as error attenuation.

PTMs can be used to derive test vectors for circuits that are susceptible to transient faults, as discussed in Krishnaswamy et al. [2005]. The general idea is to choose a multi-set of test vectors to increase the detection probability of a transient fault. For instance, in Example 2, rows with indices {001, 011, 101} deviate from the ITM the most, which indicates that they are highly sensitive to transient errors. Therefore these vectors can be repeated fewer times than less sensitive test vectors. Other applications include measuring the testability of signals under transient errors, and reliability-driven logic restructuring where parts of circuits are re-synthesized locally.

A future topic of research is exploration of ADD variants that yield further compression when representing probabilistic circuits. For instance, the use of edge-valued ADDs may curb the blowup associated with the tensor product

operation. Several BDD-variants such as indexed-BDDs [Jain et al. 1997] (where the variable ordering can be different in different subtrees), or partitioned BDDs [Jain et al. 1992] (where the signals are split into disjoint partitions) can be considered for PTM compression. Partitioned ADDs could represent partitioning the PTM into sets of conditional probabilities for subsets of input or output variables.

REFERENCES

ALEXANDRESCU, D., ANGHEL, L., AND NICOLAIDIS, M. 2002. New methods for evaluating the impact of single event transients in VDSM ICs. In *Proceedings of the IEEE Symposium on Defect and Fault Tolerance in VLSI Systems.* 99–107.

BAHAR, R. I., MUNDY, J., AND. CHAN, J. 2003. A probabilistic-based design methodology for nanoscale computation. In *Proceedings of the International Conference on Computer-Aided Design*. 480–486.

BAHAR, R. I., FROHM, E. A., GAONA, C. M., HACHTEL, G. D., MACII, E., PARDO, A. AND SOMENZI, F. 1993. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on Computer-Aided Design*. 188–191.

BRGLEZ, F., POWNALL, P., AND HUM, R. 1984. Applications of testability analysis: From ATPG to critical delay path tracing. In *Proceedings of the International Test Conference*. 705–712.

BRYANT, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput. 35*, 677–691.

CHOW, C. AND LIU, C. 1968. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inform. Theo. 14*, 11, 462–467.

CLARKE, E., FUJITA, M., AND ZHAO, X. 1996. Multi-terminal binary decision diagrams and hybrid decision diagrams. In *Representations of Discrete Functions*, T. Sasao and M. Fujita, Eds., Kluwer Academic Publishers, 93–108.

CORMEN, T., LIESERSON, C., RIVEST, R., AND STEIN, C. 2001. *Introduction to Algorithms.* MIT Press, 331–338.

DHILLON, Y. S., DIRIL, A. U., AND CHATTERJEE, A. 2005. Soft-error tolerance analysis and optimization of nanometer circuits. In *Proceedings of the Conference on Design and Test in Europe*. 288–293.

EGNER, S., PÜSCHEL, M., AND BETH, T. 1997. Decomposing a permutation into a conjugated tensor product. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*. 101–108.

ERCOLANI, S., FAVALLI, M., DAMIANI, M., OLIVIO, P., AND RICO, B. 1989. Estimate of signal probability in combinational logic networks. In *Proceedings of the European Test Conference.* 132–38.

HACHTEL, G. AND SOMENZI, F. 1996. *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers, Boston, MA.

HAN, J. AND. JONKER, P. 2002. A system architecture solution for unreliable nanoelectronic devices. *IEEE Trans. Nanotech. 1*, 201–208.

HINTON, A. AND KWIATKOWSKA, M. 2006. PRISM: A tool for automatic verification of probabilistic systems. Lecture Notes in Computer Science. vol. 3920, 441–444.

JAIN, J., BITNER, J., FUSSELL, D. S., AND ABRAHAM, J. A. 1992. Functional partitioning for verification and related problems. In *Proceedings of the Brown MIT VLSI Conference*. 210–226.

JAIN, J., BITNER, J., ABADIR, M. S., ABRAHAM, J. A., AND FUSSEL, D. S. 1997. Indexed BDDs: Algorithmic advances in techniques to represent and verify Boolean functions. *IEEE Trans. Comput. 46*, 11, 1230–1245.

KRISHNASWAMY, S., MARKOV, I. L., AND HAYES, J. P. 2005. Testing logic circuits for transient faults. In *Proceedings of the European Test Symposium.* 102–107.

KRISHNASWAMY, S., VIAMONTES, G. F., MARKOV, I. L., AND HAYES, J. P. 2005. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Proceedings of the Conference on Design Automation and Test in Europe*. 282–287.

S. KULLBACK, S. AND LEIBLER, R. A. 1951. On information and sufficiency. *Annals of Math. Stat. 22*, 1, 79–86.

LEVIN, V. L. 1964. Probability analysis of combination systems and their reliability. *Engin. Cybern. 6*, 78–84.

MALVESTUTO, F. M. 1991. Approximating discrete probability distributions with decomposable models. *IEEE Trans. Syst. Man, Cybern. 21*, 5, 1287–1894.

MISKOV-ZIVANOV, N. AND MARCULESCU, D. 2006. MARS-C: Modeling and reduction of soft errors in combinational circuits. In *Proceedings of the Design Automation Conference*. 767–772.

MOHANRAM, K. AND TOUBA, N. A. 2003. Cost-effective approach for reducing soft error failure rate in logic circuits. In *Proceedings of the International Test Conference*. 893–901.

MOHANRAM, K. 2005. Simulation of transients caused by single-event upsets in combinational logic. In *Proceedings of the International Test Conference.*

NORMAN, G., PARKER, D., KWIATKOWSKA, M., AND SHUKLA, S. 2005. Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Sys*. *24*, 10, 1629–1637.

OMANA, M., PAPASSO, G., ROSSI, D., AND METRA, C. 2003. A model for transient fault propagation in combinatorial logic. In *Proceedings of the International Online Testing Symposium*. 111–115.

PARKER, K. P. AND MCCLUSKEY, E. J. 1975. Probabilistic treatment of general combinational networks. *IEEE Trans. Comput. 24*, 6, 668–670.

PATEL, K. N., HAYES, J. P., AND MARKOV, I. L. 2003. Evaluating circuit reliability under probabilistic gate-level fault models. In *Proceedings of the International Workshop on Logic and Synthesis.* 59–64.

PIPPENGER, N. 1998. Reliable computation by formulas in the presence of noise. *IEEE Trans. Inform. Theo*. *34*, 2, 194–197.

SAVIR, J., DITLOW, G., AND BARDELL, P. H. 1983. Random pattern testability. In *Proceedings of the IEEE Symposium on Fault Tolerant Computing*. 80–89.

SHIVAKUMAR, P., KISTLER, M., KECKLER, S. W., BURGER, D., AND ALVISI, L. 2002. Modeling the effect of technology trends on soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks.* 389–398.

SRINIVASAN, R., GUPTA, S. K., AND BREUER, M. A. 1993. An efficient partitioning strategy for pseudo-exhaustive testing. In *Proceedings of the Design Automation Conference*. 242–248.

VIAMONTES, G. F., MARKOV, I. L., AND HAYES, J. P. 2003. Improving gate-level simulation of quantum circuits. *Quant. Inform. Proces*. *2*, 5, 347–380.

VIAMONTES, G. F., RAJAGOPALAN, M., MARKOV, I. L., AND HAYES, J. P. 2003. Gate-level simulation of quantum circuits. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 295–301.

VON NEUMANN, J. 1956. Probabilistic logics and synthesis of reliable organisms from unreliable components. In *Automata Studies,* C.E. Shannon and J. McCarthy Eds., Princeton University Press, 43–98.

ZHANG, B., WANG, W. S., AND ORSHANSKY, M. 2006. FASER: Fast analysis of soft error susceptibility for cell-based designs. In *Proceedings of the International Symposim on Quality Electronic Design*. 755–760.

ZHANG, M. AND SHANBHAG, N. R. 2004. A soft error rate analysis (SERA) methodology. In *Proceedings of the International Conference on Computer Aided Design*. 111–118.

ZHAO, C., BAI, X., AND DEY, S. 2004. A scalable soft spot analysis methodology for compound noise effects in nano-meter circuits. In *Proceedings of the Design Automation Conference*. 894–899.

Received June 2006; revised March 2007, June 2007; accepted July 2007