Mutually Disjoint Signals and Probability Calculation in Digital Circuits

Vishwani D. Agrawal
Bell Labs, Lucent Technologies
700 Mountain Avenue
Murray Hill, NJ 07974
va@research.bell-labs.com

Sharad Seth
University of Nebraska
Dept. of Computer Science
Lincoln, NE 68588
seth@cse.unl.edu

Abstract

Signal probability calculation in circuits where signals are not independent is generally expensive. We show that some correlated signals may be mutually disjoint. In such cases, the probability calculation can be as simple as it is for independent signals. For example, two signals that cannot be simultaneously true are defined as OR-disjoint. If these signals feed an OR gate, the probability of the output being true is simply the sum of the probabilities of inputs being true. We give an implication-based algorithm for identifying disjoint signals. Examples of large adders illustrate how the identification of disjoint signals simplifies the probability calculation.

1 Introduction

When random input patterns are applied to a combinational circuit, with each signal, we can associate a probability of assuming the "one" value. Signal probabilities of digital circuits have several applications like estimation of testability [1] and power dissipation [2]. Their exact calculation is complex and various approximations are used. Approximations normally amount to assuming signals as statistically independent when, in fact, they are not so. The idea of the present work is to give an alternative way to calculate signal probabilities from mutually disjoint signals. *A set of digital signals is mutually disjoint if at most one signal can be* true *at any given time.* Such signals are always statistically dependent. Yet, in many practical situations disjointness of signals can be readily established and our formula can compute exact signal probabilities.

Let us first summarize the known case of *independent* signals. We will denote a Boolean variable by an upper case letter and its probability of assuming a value 1 by the corresponding lower case letter. Consider an AND gate $C = AB$. If A and B are independent, then

$$c_{AND} = ab \tag{1}$$

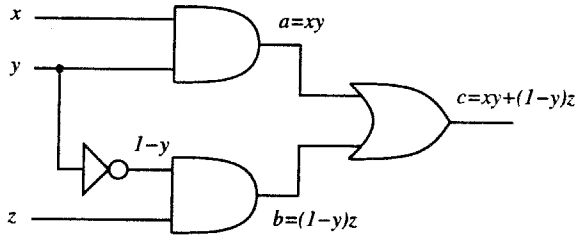

Figure 1: A multiplexer circuit

This formula is exact and is used to compute signal probabilities for other gates as well. For example, consider an OR gate $C = A + B$. From de Morgan's law, $\overline{C} = \overline{A}\,\overline{B}$. Also, inversion does not alter the independence property. That is, if two signals are mutually independent, then their inversions will also be independent. For independent A and B, we use the AND formula of Equation 1 to obtain,

$$1 - c_{OR} = (1 - a)(1 - b)$$

and, therefore,

$$c_{OR} = a + b - ab \tag{2}$$

The *AND formulas* of Equations 1 and 2 can be used without difficulty whenever signals are independent. However, reconvergent fanouts make signals dependent on each other. For example, consider the multiplexer circuit in Figure 1. Both A and B depend on Y. If we applied Equation 2 to compute c, ignoring signal dependence, we get:

$$\begin{aligned} c = a + b - ab &= xy + (1 - y)z - xy(1 - y)z \\ &= xy + z - yz - xyz + xy^2z \end{aligned} \tag{3}$$

Parker and McCluskey [3] have pointed out that this approach would still work with a proper interpretation of the powers of a probability variable. For example, in Equation 3, y^2 represents probability of two instances

0-8186-8409-7/98 $10.00 © 1998 IEEE

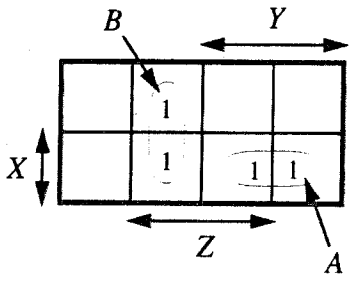

Figure 2: Karnaugh map showing disjoint A and B

of the signal y being *true*, which is the same as the probability that y is *true*. Thus, we obtain

$$c = xy + z - yz \tag{4}$$

This is the correct probability for the output. The preceding calculation shows that whenever signals are correlated, one must obtain an algebraic polynomial for the probability in terms of the probabilities of independent primary inputs. For large circuits, such expressions can become unmanageable. Sometimes, the circuit can be partitioned in such a way that the inputs to a partition are independent signals [1]. These partitions, known as *supergates*, can also become too large in extreme cases. Another method is to use binary decision diagrams (BDDs) [2, 4]. This requires construction of a BDD for every signal, but the technique can be applied where the size of BDDs is manageable.

2 New Result

We will derive an alternative exact formula for probability calculation. Consider the circuit of Figure 1 again. Since inputs, X, Y and Z, are independent variables, exact probabilities for A and B are obtained from the AND formula. These are shown in Figure 1. Notice in the Karnaugh map of the circuit (Figure 2) that A and B are mutually disjoint signals. Since the OR gate provides the *union* of logic events, the probability of C is $c = a + b = xy + z - yz$.

Thus, our new result is for an OR gate $C = A + B$: If A and B are disjoint (in fact, disjoint events are mutually dependent) signals, then

$$c_{OR} = a + b \tag{5}$$

is the exact probability of the output. Comparing the formula with Equation 2, we note that, in general, the signal probability at the output of an OR gate is larger for disjoint than for independent input signals. From duality, it can be shown that the opposite is true for an AND gate. Also, inversions that preserve independence, in fact, alter the disjointness property. The OR formula can be applied to an AND gate, via the De

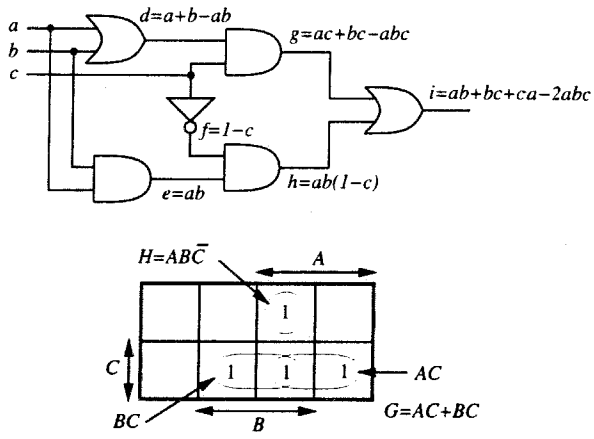

Figure 3: A carry circuit designed using a multiplexer

Morgan's law, only when $\overline{A}$ and $\overline{B}$ are disjoint. Then, for an AND gate $C = AB = \overline{\overline{A} + \overline{B}}$, we can compute

$$c_{AND} = 1 - (1 - a + 1 - b) = a + b - 1 \tag{6}$$

We refer to Equations 5 and 6 as *disjoint formulas*. An important observation about the previous example (Figure 1) is that Equation 5 holds for C even when X and Z are not independent. We do require that Y should be independent of X and Y. This is because the inputs to the OR gate remain disjoint due to Y and $\overline{Y}$. Thus we can extend the disjoint formula to mux-based implementations[1] of commonly used functions. As an illustration, consider the implementation of the *carry* function shown in Figure 3. The Boolean functions of the two data inputs to the multiplexer are, respectively, $A + B$ and AB, which are neither independent nor disjoint. However, the two input signals for the final OR gate are still disjoint (as is evident from the Karnaugh map) and hence the disjoint formula is applicable.

3 Determination of Disjointness

When input signals of a gate are disjoint, the output signal probability is numerically calculated directly from input probabilities. A similar property holds when inputs are independent. Thus, if in a circuit all gates can be classified as having inputs that are either disjoint or independent, then signal probabilities can be computed very easily. Independence of signals requires that they should not depend on common primary inputs. An efficient labeling algorithm can be devised for the circuit graph to establish independence of signals [1, 5]. We will now develop an algorithm for disjointness.

Definition 1: In a set of *OR-disjoint signals* at most one signal can be *true* at any time.

[1] Mux-based implementations are efficient in CMOS and are commonly employed in the design of FPGA cells.

```
Set a logic value v = 1 if G is OR or NOR, else v = 0
For each input of G
    Set input to v and perform implications
    If any input of G implied to v
        Exit with "G not disjoint"
    Else
        If any input of G is unknown
            Exit with "status of G undetermined"
        End if
    End if
End for
Normal exit: "G is disjoint"
```

Figure 4: Algorithm for classifying a gate G as disjoint

Clearly, the output probability of an OR or NOR gate with OR-disjoint inputs is directly obtained as the sum of input signal probabilities (see Section 2).

Property 1: If two signals are OR-disjoint, then a 11 pattern on them is impossible (this follows from Definition 1.)

Definition 2: In a set of *AND-disjoint signals* only one signal can be *false* at any time.

According to the results of Section 2, if an AND or NAND gate has AND-disjoint inputs, then the output probability is directly computed from input probabilities.

Property 2: If two signals are AND-disjoint, then a 00 pattern on them is impossible (this follows from Definition 2.)

Definition 3: We call an OR or a NOR gate *disjoint* if its inputs are OR-disjoint. Similarly, an AND or a NAND gate will be called *disjoint* if its inputs are AND-disjoint.

To determine whether a gate is disjoint, we will use an implication procedure. In this procedure, a set of signals is set to some specified values and all backward and forward implications are carried out. It is a simulation-like procedure that completes in linear time. It has previously been used in redundancy identification [6]. The algorithm of Figure 4 can in many cases determine if a gate G is disjoint. Depending upon the gate-type, a logic value is placed on one input of G and implications are carried out. For example, if G is an OR gate, we will set one of its inputs to 1. For G to be disjoint, the implication should set all other inputs of G to 0. This procedure is repeated for each input of G. Specifically for two-input gates, a simpler algorithm can be derived using properties 1 and 2. In that case, for an OR gate, we will place a 11 pattern on inputs and carry out implications. If we find a *contradiction*, i.e., 11 pattern is impossible then G is disjoint. Contradiction analysis can be very efficient and has been used for test generation and redundancy identification [7]. The algorithm of Figure 4 relies on finding implications. Direct (or *local*) implications are easy and can be found

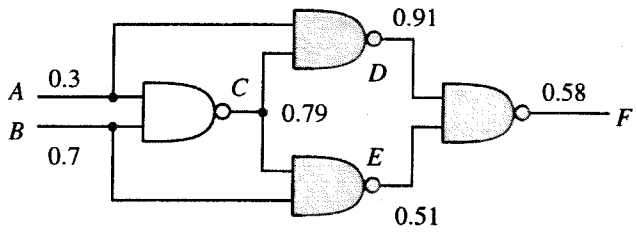

Figure 5: A NAND implementation of exclusive-OR

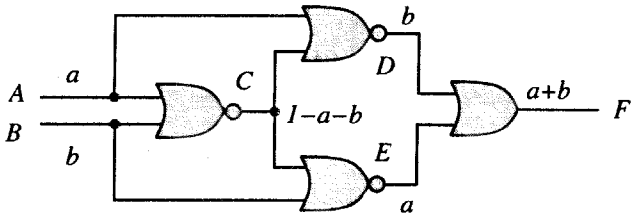

Figure 6: A NOR/OR implementation of exclusive-OR

in linear time. However, such procedures will not find all implications. Global implications require symbolic analysis and have higher time and memory complexities. Efficient techniques for global implications are discussed by Kunz and Stoffel [8].

In the example of Figure 5, all shaded NAND gates are disjoint. The unshaded gate has independent inputs. Thus probability calculation is straightforward. We illustrate the calculation for input probabilities $a = 0.3$ and $b = 0.7$. For gate C, inputs are independent and $c = 1 - ab = 0.79$. For gate D, inputs are disjoint and $d = (1 - a) + (1 - c) = 0.91$. Similarly, gates E and F are disjoint and have output probabilities, $e = (1 - b) + (1 - c) = 0.51$ and $f = (1 - d) + (1 - e) = 0.58$, respectively. While this illustration shows the calculation of probabilities of signals inside an exclusive-OR block, for independent inputs A and B, the output probability can be found by algebraic substitutions as:

$$f_{EOR} = a + b - 2ab \tag{7}$$

Next, suppose the inputs A and B of an exclusive-OR block were OR-disjoint. Considering the NOR/OR implementation of Figure 6, we find that all four gates are OR-disjoint. Thus, for OR-disjoint inputs the output probability of an exclusive-OR block is simply given by,

$$f_{EOR} = a + b \tag{8}$$

Figure 7 shows the probability calculation for a full adder circuit with input probabilities $a = 0.2$, $b = 0.3$ and $c = 0.1$. The gates found disjoint by the algorithm of Figure 4 are shown in gray. The two unshaded gates

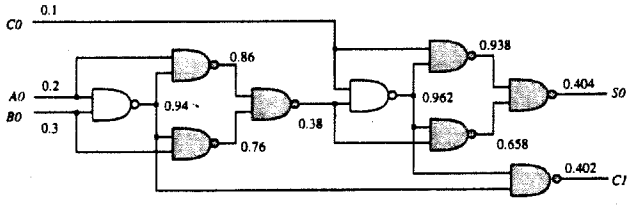

Figure 7: Probability calculation for a full-adder circuit

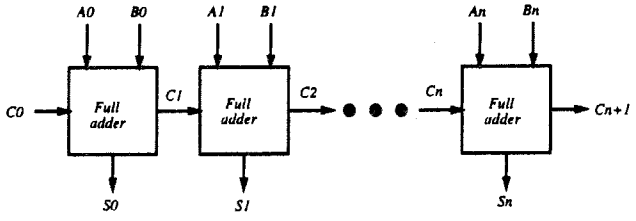

Figure 8: A ripple-carry adder

have independent inputs. Once the gates are identified as "independent" or "disjoint", probability calculation is done in one pass from left to right. For every gate, whose input signal probabilities have been calculated, the output probability is computed directly as explained in the previous paragraph.

4 Large Circuits

Such a calculation may be possible in arbitrarily large circuits also. Consider the ripple-carry adder of Figure 8. Suppose each full-adder block is implemented by a circuit as shown in Figure 7. Now all but two gates in each full-adder block will still be classified as disjoint by the algorithm of Figure 4. The remaining two gates have independent inputs. Thus, for any given input probability values of $A0, B0, C0, ..., An, Bn$, probabilities for outputs $S0, ..., Sn, Cn+1$, as well as for all internal signals are calculated in one pass.

The ripple-carry adder is a special case of the circuits that can be partitioned into blocks (full-adders in this case) that fully contain all the supergates (partitions with mutually independent inputs) [5] in the circuit. The ripple-carry adder is also a special case of a K-bounded circuit [9] that can be partitioned into blocks enclosing all reconvergences and have a fan-in of at most K. In either class of circuits, signal probabilities for each block can be computed independently. The computation for each block, however, could be difficult if blocks are large or have complex signal dependencies. The full-adder example shows that the disjoint computation model may greatly simplify probability computation in such cases.

Consider the carry look-ahead adder of Figure 9 as discussed by Becker [10]. The modules gp, GP and S

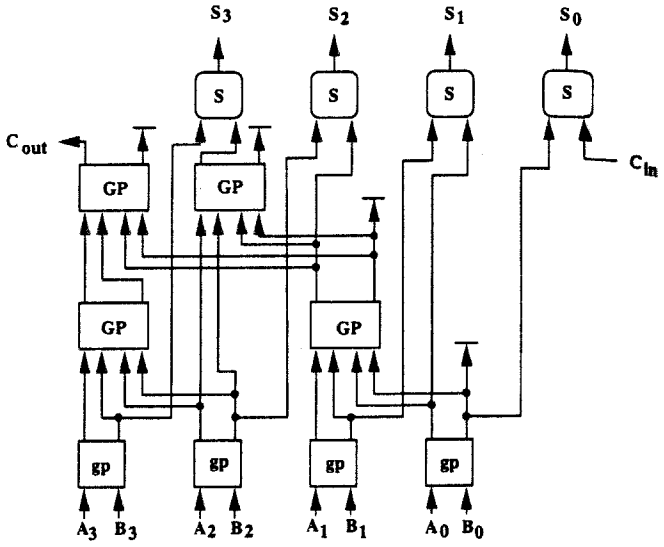

Figure 9: A carry look-ahead adder

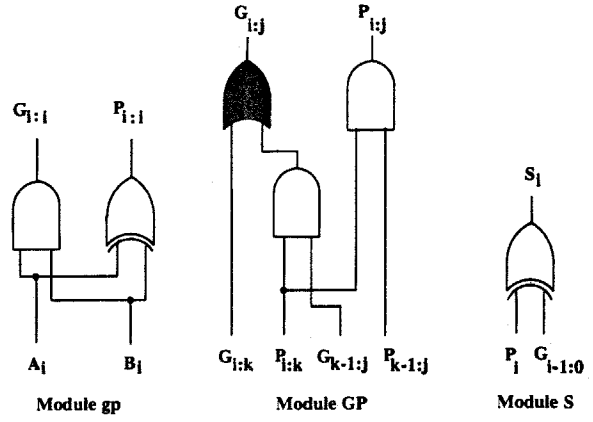

Figure 10: Modules used in Figure 9

are shown in Figure 10. We assume that the inputs to the adder are independent signals. Reconvergences exist only within the modules. All gp modules have independent inputs and their output probabilities are calculated by Equations 1 and 7. We notice that the two outputs of a gp module contain no common minterms. These are, therefore, OR-disjoint.

Consider a GP module. It has two pairs of inputs. The signals in one pair are independent of the signals in the second pair. If we assume that the signals within each pair are OR-disjoint, then the inputs of the OR gate (shown shaded) are disjoint. Also, the two outputs of the GP module are found to be OR-disjoint. Probabilities for both AND gates are calculated by Equation 1 and that for the OR gate by Equation 6. The output probability for each S module is calculated by Equation 7 since its inputs are independent.

The above analysis of the three modules allows us

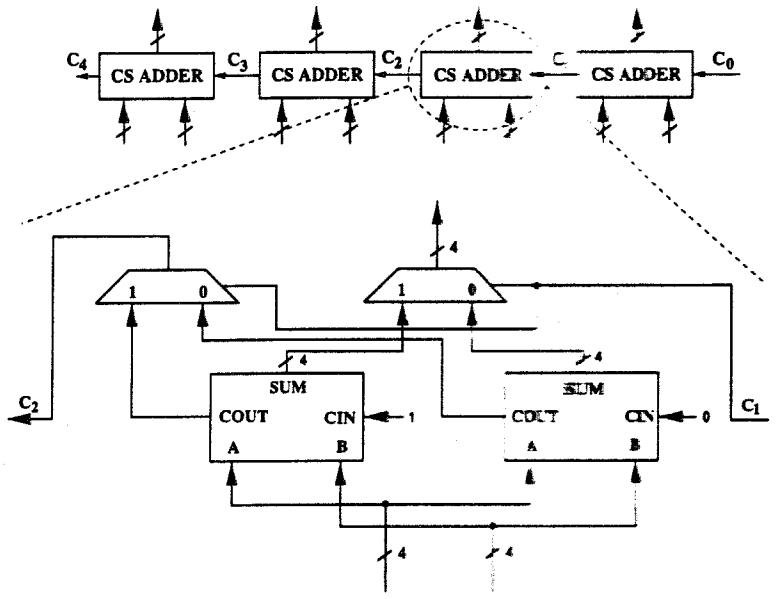

Figure 11: A carry-select adder

o characterize each gate in the circuit of Figure 9 as ither independent or disjoint. Thus, probabilities for ll signals can be locally calculated.

Next, consider a carry-select adder as shown in Fig- re 11 [11]. It contains an array of carry-select adders ith a carry signal flowing from right to left. Figure 11 ives an expanded view of an adder in the array. It ontains two SUM blocks, each of which can be either f the ripple-carry or carry look-ahead type. Probabil- ies for all signals in these blocks can be computed as iscussed before. The outputs SUM and COUT pro- uced by these blocks are not independent. However, ese are combined in multiplexers, using an indepen- nt signal C_1 generated by the previous stage. C_1 is ndependent of the data signals of multiplexers. Ac- ording to the analysis of Section 2, all gates in mul- plexers can be classified as either independent or dis- int. Thus, the probabilities for all gates, including ose producing adder outputs and C_2 can be directly lculated.

Another type of adder, known as the conditional- m adder, is similar to the carry-select adder in that ch stage computes SUM and $SUM + 1$, indepen- ntly for a bit position. A multiplexer tree is used to nerate the final result. It can be verified that such a nditional-sum adder [10] is also amenable to the sim- e signal probability calculation using the independent d disjoint properties.

Other Applications and Remarks

We should point out that the ease of computation pends on the circuit structure. Figure 12 shows an-

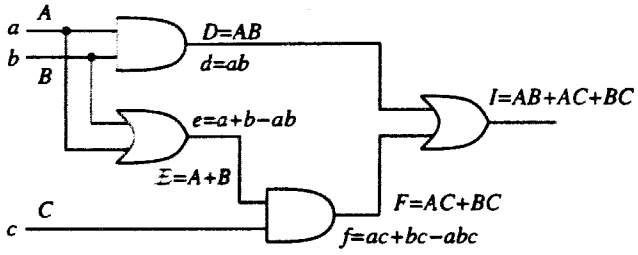

Figure 12: An alternative circuit for carry function

other implementation of the carry function. Here probabilities for D, E and F are obtained by independence formulas. However, the two inputs of the last OR gate, D and F, are neither independent (since both involve common signals A and B) nor disjoint (since they contain a common minterm ABC; see Figure 3). Thus, an exact calculation of signal probability for I will require a more complex analysis of the previously published methods [1, 8].

There are known methods that determine whether or not a set of internal signals is a mutually independent set. Similarly, algorithms, such as the one shown in Figure 4, can be effective in identifying mutually disjoint sets. Another method is to verify that the functional cubes (not necessarily minimized) of the signals are disjoint. A local analysis of the circuit can also establish disjointness. As an example, notice that the signals at the output of the two AND gates in Figure 1 remain disjoint even if the three primary input signals were derived from an arbitrary combinational logic block. Other algorithms may be possible.

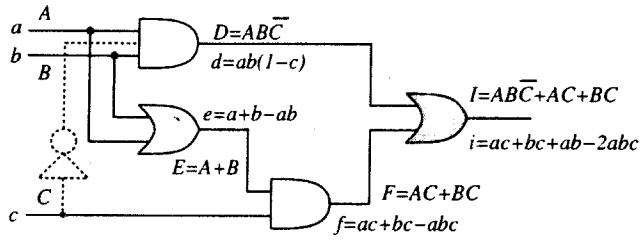

Figure 13: A computation model for circuit of Figure 12

Sometimes, local changes in the circuit can be applied to facilitate probability calculation. In Figure 13 an inverter is added to the carry circuit of Figure 12. This makes signals D and F disjoint without changing the function and the probability of I is obtained according to Equation 6. The inverter here carries a primary input (signal C) and has been added to create a computation model. Using both independence formula and the disjoint formula, appropriately where one is applicable, can simplify probability calculation for a circuit.

One can also use disjoint cube extraction (there are fast PODEM-based algorithms) to compute signal probability. Since each cube is a product of primary input literals and the primary inputs are assumed to be independent, the probability of the cube is obtained by the AND formula of Equation 1. The cube probabilities are then summed according to the disjoint formula of Equation 6. This procedure has been applied to the characteristic polynomial method of logic verification [12, 13]. In that method, two logic functions are found identical if, for randomly selected input signal probabilities, the output probabilities coincide. The reported results show that exact output probabilities for all 17 outputs of two 16-bit adders (one a truth table specification and the other a 197-gate multilevel implementation) were calculated in 0.5 second on a Sun Sparc 2 workstation. This adder was implemented for optimized area and delay and used neither the ripple-carry architecture of Figure 8 nor the carry look-ahead or carry-select schemes of Figures 9 and 11.

6 Conclusion

Similar to the K-bounded circuits we mentioned in Section 4, there are other generalized structures with polynomial-time test generation complexity [14]. Since the exact signal probability calculation has a similar complexity as test generation, it would be interesting to explore such circuits. Another application of the disjoint property might be in logic design. For example, in a two-level implementation, if one selects cubes that are nonoverlapping, then probabilities of all signals can be directly computed by independence and disjoint formulas. The design, however, will be costlier since it may require smaller, and sometimes more, cubes. Specific testability advantage of such circuits is a topic yet to be fully explored.

References

[1] S. C. Seth and V. D. Agrawal, "A New Model for Computation of Probabilistic Testability in Combinational Circuits," *Integration, the VLSI Journal*, vol. 7, pp. 49–75, 1989.

[2] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits," *IEEE Trans. CAD*, vol. 12, pp. 310–323, February 1993. See Appendix B.

[3] K. P. Parker and E. J. McCluskey, "Proabilistic Treatment of General Combinational Networks," *IEEE Trans. Computers*, vol. C-24, pp. 668–670, June 1975.

[4] R. Krieger, B. Becker, and R. Sinković, "A BDD-based Algorithm for Computation of Exact Fault Detection Probabilities," in *Proc. 23rd Fault Tolerant Computing Symp.*, pp. 186–195, 1993.

[5] S. C. Seth, B. B. Bhattacharya, and V. D. Agrawal, "An Exact Analysis for Efficient Computation of Random-Pattern Testability in Combinational Circuits," in *Proc. 16th Fault Tolerant Computing Symp.*, pp. 318–323, 1986.

[6] M. A. Iyer and M. Abramovici, "Low-Cost Redundancy Identification for Combinational Circuits," in *Proc. 7th International Conf. VLSI Design*, pp. 315–318, 1994.

[7] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler, "A Transitive Closure Algorithm for Test Generation," *IEEE Trans. CAD*, vol. 12, pp. 1015–1028, July 1993.

[8] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks; Logic Synthesis and Verification Using Testing Techniques*. Boston: Kluwer Academic, 1997.

[9] H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problem for Combinational Circuits," *IEEE Trans. Computers*, vol. C-31, pp. 555–560, June 1982.

[10] B. Becker, "Efficient Testing of Optimal Time Adders," *IEEE Trans. Computers*, vol. 37, pp. 1113–1121, September 1988.

[11] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. Reading, Mass.: Addison-Wesley, second ed., 1993. p. 532.

[12] V. D. Agrawal and D. Lee, "Characteristic Polynomial Method for Verification and Test of Combinational Circuits," in *Proc. 9th International Conf. VLSI Design*, pp. 341–342, January 1996.

[13] V. D. Agrawal, S. L. Grant, D. Lee, and H. Woźniakowski, "A Software Systemm for Logic Verification Using the Characteristic Polynomial Method," in *Proc. 14th Lucent Conf. on Electronic Testing*, (Princeton), pp. 183–188, April 1997.

[14] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell, "Energy Minimization and Design for Testability," *Journal of Electronic Testing: Theory and Applications*, vol. 5, pp. 57–66, February 1994.