# TECHNIQUES FOR ESTIMATION OF DESIGN DIVERSITY FOR COMBINATIONAL LOGIC CIRCUITS

Subhasish Mitra, Nirmal R. Saxena and Edward J. McCluskey
Center for Reliable Computing
Departments of Electrical Engineering and Computer Science
Stanford University, Stanford, California
http://crc.stanford.edu

**Abstract**

*Design diversity has long been used to protect redundant systems against common-mode failures. The conventional notion of diversity relies on "independent" generation of "different" implementations of the same logic function. This concept is qualitative and does not provide a basis to compare the reliabilities of two diverse systems. In a recent paper, we presented a metric to quantify diversity among several designs. The problem of calculating the diversity metric is NP-complete and can be of exponential complexity. In this paper, we present techniques to estimate the value of the design diversity metric. For datapath designs, we have formulated very fast techniques to calculate the value of the metric by exploiting the regularity in the datapath structures. For general combinational logic circuits, we present an adaptive Monte-Carlo simulation technique for estimating bounds on the value of the metric. The adaptive Monte-Carlo simulation technique provides accurate estimates of the design diversity metric; the number of simulations used to reach this estimate is polynomial (instead of exponential) in the number of circuit inputs. Moreover, the number of simulations can be tuned depending on the desired accuracy.*

## 1. Introduction

Concurrent Error Detection (CED) techniques are widely used for designing systems with high data integrity. A duplex system is an example of a classical redundancy scheme that has been used in the past for concurrent error detection. There are many examples of commercial dependable systems from companies like Stratus and Sequoia using hardware duplication [Kraft 81, Pradhan 96]. Hardware duplication is also used in the IBM G5 processor [Webb 97, Spainhower 99] and also in the space shuttle. Figure 1.1 shows the basic structure of a duplex system.

In a duplex system there are two modules (shown in Fig. 1.1 as Module 1 and Module 2) that implement the same or related logic functions (e.g., complement function). The two implementations can be the same or different. A comparator is used to check whether the outputs from the two modules agree. If the outputs disagree, the system indicates the presence of an error. *Data integrity* means that the system either produces correct outputs or generates error signal when incorrect outputs are produced. For a duplex system, data integrity is maintained as long as both modules do not produce identical erroneous outputs.

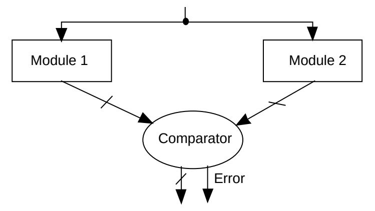

Figure 1.1. A Duplex System

In a duplex system *common-mode failures* (CMFs) result from failures that affect more than one element at the same time, generally due to a single cause [Lala 94]. These include operational failures that may be due to external (such as EMI, power-supply disturbances and radiation) or internal causes and design faults. Common-mode failures in redundant VLSI systems are surveyed in [Mitra 00a]. *Design diversity* has been proposed in the past to protect redundant systems against common-mode failures. In [Avizienis 84], *design diversity* was defined as the "independent" generation of two or more software or hardware elements (e.g., program modules, VLSI circuit masks, etc.) to satisfy a given requirement. Design diversity has been applied to both software and hardware systems [Lyu 91, Briere 93, Riter 95]. Tohma proposed using the implementations of logic functions in true and complemented forms during duplication [Tohma 71]. The use of a particular circuit and its dual was proposed in [Tamir 84] to achieve diversity in order to handle common-mode failures. The basic idea is that, with different implementations, common failure modes will probably cause different error effects.

The above discussion of diversity is qualitative and does not provide any quantitative insight into the design or the analysis of systems using diverse duplication. In a recent paper [Mitra 99a], we developed a metric (called the D-metric) to quantify diversity among several designs and used this metric to perform reliability analysis of redundant systems. However, for arbitrary designs, the problem of calculating the value of the D-metric is NP-complete. In this paper, we present several techniques to calculate the D-metric. The contributions of this paper are: (1) A technique to reduce the list of fault pairs in duplex systems (of combinational logic circuits) that must be considered while calculating the value of the D-metric. The technique significantly reduces the list of fault pairs as demonstrated by simulation results; (2) Fast techniques for computing the D-metric for datapath logic circuits by exploiting the regularity of the datapath structures; (3) Modeling the diversity calculation problem as a signal probability calculation problem for general combinational logic circuits; and, (4) An adaptive Monte-Carlo simulation technique for estimating the value of the D-metric for general combinational logic circuits using orders of magnitude fewer simulations compared to exhaustive simulation.

In Sec. 2, we present the D-metric that was first introduced in [Mitra 99a]. In Sec. 3 we prove that the problem of calculating the D-metric is an NP-complete problem. Then we formulate techniques to reduce the number of fault pairs to be considered. We also present simulation results demonstrating the effectiveness of these reduction techniques. Section 4 describes techniques to calculate the D-metric for datapath circuits like adders. In Sec. 5, we consider general combinational logic circuits and model the problem of calculating the D-metric as a signal probability calculation problem. We describe an adaptive Monte-Carlo simulation technique to calculate signal probability and present results to demonstrate the effectiveness of this simulation technique. Finally, we conclude in Sec. 6.

## 2. *D*: A Design Diversity Metric

Assume that we are given two implementations (logic networks) of a logic function, an input probability distribution and faults $f_i$ and $f_j$ that occur in the first and the second implementations, respectively. The *diversity* $d_{i,j}$ *with respect to the fault pair* ($f_i$, $f_j$) is the conditional probability that the two implementations do not produce identical errors, given that faults $f_i$ and $f_j$ have occurred [Mitra 99a].

The $d_{i,j}$'s generate a diversity profile for the two implementations with respect to a fault model. Consider a duplex system consisting of the two implementations under consideration. In response to any input combination, the implementations can produce one of the following cases at their outputs: (1) Both of them produce correct outputs. (2) One of them produces the correct output and the other produces an incorrect output. (3) Both of them produce the same incorrect value. (4) They produce non-identical incorrect outputs.

For the first case, the duplex system will produce correct outputs. For the second and the fourth cases, the system will report a mismatch so that appropriate recovery actions can be taken. However, for the third case, the system will produce an incorrect output without reporting a mismatch — thus, for the third case, the data integrity of the system is not preserved.

For a given fault model, the *design diversity metric*, *D*, of two designs is the expected value of the diversity with respect to different fault pairs. Mathematically, we have $D = \sum_{(f_i, f_j)} P(f_i, f_j) d_{i,j}$, where $P$ ($f_i$, $f_j$) is the probability of the fault pair ($f_i$, $f_j$).

*D* is the probability that the two implementations either produce error-free outputs or produce *different* error patterns on their outputs in the presence of faults affecting the two implementations.

Consider any combinational logic function with *n* inputs and a single output. The single stuck-at fault model is used because of its effectiveness as discussed in [McCluskey 00]. Consider two implementations ($N_1$ and $N_2$) of the given combinational logic function with faults pair $f_i$ and $f_j$ affecting $N_1$ and $N_2$, respectively.

The *joint detectability*, $k_{i,j}$, of a fault pair ($f_i$, $f_j$) is the number of input patterns that detect both $f_i$ and $f_j$. This definition follows from the idea of detectability developed in [McCluskey 88].

Assuming all input patterns are equally likely, $d_{i,j} = 1 - \frac{k_{i,j}}{2^n}$.

For example, consider the two implementations of the logic function $Z = AB + AC$ shown in Fig. 2.1.

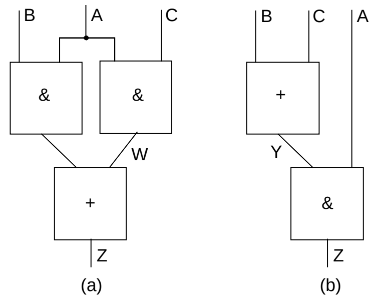

Figure 2.1. Example of diversity

Consider the fault $f_1$ = *w* stuck-at-0 in the implementation of Fig. 2.1a and the fault $f_2$ = *y* stuck-at-0 in the implementation of Fig. 2.1b. The set of input combinations that detect $f_1$ is {ABC = 101}. The set of input combinations that detect $f_2$ is {ABC = 111, 101, 110}. It is clear that ABC = 101 is the only input combination that detects both $f_1$ and $f_2$. Hence, the joint detectability $k_{1,2}$ of the fault pair ($f_1$, $f_2$) is 1. If a duplex system consisting of the two implementations in Fig. 2.1 is affected by the fault pair ($f_1$, $f_2$), then ABC = 101 is the only input combination for which both implementations will produce identical errors. If we assume that all input combinations are equally likely, then the $d_{1,2}$ for the fault pair ($f_1$, $f_2$) is $1 - \frac{1}{8} = \frac{7}{8}$.

Assuming that all fault pairs are equally probable and that there are $m$ fault pairs ($f_i$, $f_j$) (i.e., $P(f_i, f_j) = \frac{1}{m}$), then the $D$ metric for the two implementations is:

$$D = \frac{1}{m}\sum_{i,j} d_{i,j}\ .$$

We extend the above example to multiple-output combinational logic circuits. *For a fault pair ($f_i$ $f_j$) affecting the two implementations, we define $k_{i,j}$ as the number of input patterns, in response to each of which, both the implementations produce the same erroneous output pattern.* Now, we can use the same formulas as in the single output case.

Table 2.1. Behavior of faulty multiple-output circuits

| Inputs | Fault-free outputs | Faulty outputs ($N_1$) | Faulty outputs ($N_2$) |
|---|---|---|---|
| 00 | 0 1 | 0 **0** | **1 0** |
| 01 | 1 0 | 1 0 | 1 0 |
| 10 | 0 0 | **1** 0 | **1** 0 |
| 11 | 1 1 | 1 **0** | 1 **0** |

For example, consider a combinational logic function with two inputs and two outputs (Table 2.1). Suppose that faults $f_i$ and $f_j$ affect the first and the second implementations, respectively. The responses of the two implementations in the presence of the faults are shown in Table 2.1. The faulty output bits are highlighted in the third and fourth columns of Table 2.1. It is clear that $k_{i,j}$ = 2 (since the implementations produce identical errors in response to input combinations 10 and 11) and the value of $d_{i,j}$ is $\frac{2}{4} = 0.5$.

The *D*-metric can be used to perform data integrity analysis of duplex systems [Mitra 99a]. An estimate of data corruption latency using the above metric was presented in [Mitra 99b]. Suppose that faults $f_1$ and $f_2$ affect the two implementations $N_1$ and $N_2$ at cycle $c$. The *data corruption latency* is defined to be the number of cycles from $c$ after which both the implementations produce the same error pattern at the output. The expression for the expected data corruption latency of a duplex system is shown below.

Expected data corruption latency =

$$\sum_{(f_1, f_2), d_{1,2} \neq 1} \frac{P(f_1, f_2)}{(1 - d_{1,2})} + \sum_{(f_1, f_2), d_{1,2} = 1} P(f_1, f_2)T\ .$$

In the above expression, $T$ is the mission time of the application under consideration. When the $d_{1,2}$ value of a fault pair is 1, the data corruption latency is strictly infinity (because data integrity is guaranteed) and is limited by the system mission time. It is clear from the above expression that the fault pairs having their $d_{i,j}$ values very close to 1 contribute the most to increase the data corruption latency. Fault pairs with very low values of $d_{i,j}$ have very little impact on increasing the expected data corruption latency of the system.

## 3. Techniques for Reducing the Number of Fault Pairs

In this section, we first prove that the problem of calculating the *D*-metric presented in Sec. 2 is NP-complete.

**Theorem 1:** The calculation of the $d_{i,j}$ value for a fault pair ($f_i$, $f_j$) is an NP-complete problem for arbitrary logic networks.

**Proof:** Consider any fault $f_i$ in an arbitrary combinational logic network. We want to find whether the fault is redundant or not. For that purpose, we can calculate the $d_{i,i}$ value for two identical designs (corresponding to the given combinational logic network). The fault is redundant if and only if the $d_{i,i}$ value is 1. However, we know that the problem of identification of a redundant fault is NP-complete because it can be reduced to the Boolean Satisfiability problem [Garey 79]. Hence, the problem of calculation of the $d_{i,j}$ values is NP-complete. Q.E.D.

For practical purposes, there are two problems associated with calculation of the D-metric. First, the number of fault pairs for which the $d_{i,j}$ values must be calculated can be very large. Second, the problem of calculating the $d_{i,j}$ value for a fault pair is NP-complete. In this section, we present techniques to reduce the number of fault pairs for which the $d_{i,j}$ values have to calculated – as a result, we obtain bounds on the *D*-metric for two implementations of a combinational logic function. The fault model that we consider is the single stuck-at fault model; i.e., all failures act as single stuck-at faults in $N_1$ and $N_2$. As our basis for the calculation of the lower and the upper bounds, we use the following two theorems.

**Theorem 2:** In a single-output fanout-free circuit C, for any single stuck-at fault *f*, the set of all test patterns that detect *f* is a subset of the set of all test patterns of either the stuck-at-0 or the stuck-at-1 fault at the output of C.

**Theorem 3:** In a single-output fanout-free circuit C, for any fault *f*, we can find a set S of single stuck-at faults at the inputs of C such that the set of all test patterns that detect *f* is a superset of the set of all test patterns that detect the faults in S.

The proofs of the above theorems follow directly from the analysis of the equivalence and dominance relationship of single-stuck faults in logic networks [To 73]. Consider the single-output fanout-free combinational logic network of Fig. 3.1. Consider the fault lead *r* stuck-at 0 (*r*/0). This fault dominates the stuck-at faults $d/0$, $e/0$ and $f/0$. Hence, the set of test patterns that detect $r/0$ is a superset of the set of test patterns that detect any one of $d/0$, $e/0$ and $f/0$. If $T_r$, $T_d$, $T_e$ and $T_f$ are the test sets that detect $r/0$, $d/0$, $e/0$ and $f/0$, respectively, then, we can infer $T_r \supseteq T_d \cup T_e \cup T_f$. Next, consider the fault $x/0$. This fault dominates the fault $n/0$ and is equivalent to the fault $z/0$. Thus, if $T_z$ is the set of test patterns that detect $z/0$, then $T_z \supseteq T_r$. Thus, we get the following supersets and subsets for $T_r$: $T_d \cup T_e \cup T_f \subseteq T_r \subseteq T_z$

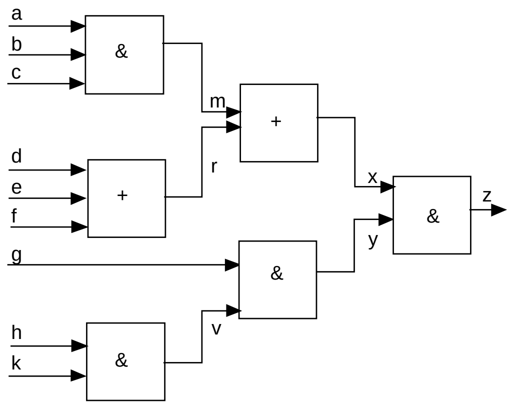

Figure 3.1. A fanout-free combinational logic network

Given any arbitrary combinational logic network (not necessarily fanout-free or single-output), we can decompose the network into maximal single-output fanout-free regions and calculate the test sets of the single stuck-at faults at the inputs and the outputs of the different fanout-free regions (Fig. 3.2). Next, we can approximate the test sets of the stuck-at faults on the remaining leads of the given network using the superset and subset relationships explained earlier and calculate bounds on the $d_{i,j}$ values of for different fault pairs as explained next.

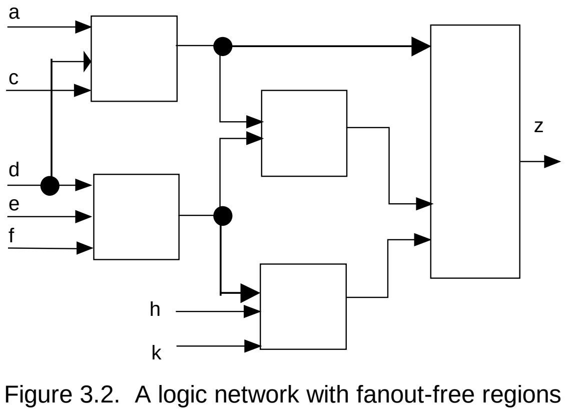

Figure 3.2. A logic network with fanout-free regions

Consider two implementations $N_1$ and $N_2$ of a single output combinational logic circuit with $n$ inputs. Let us consider two faults $f_1$ and $f_2$ affecting $N_1$ and $N_2$, respectively. Let us suppose that $T_1$ and $T_2$ are the sets of input combinations that detect $f_1$ and $f_2$, respectively. From our example in Sec. 2, we know that $d_{1,2} = 1 - \frac{|T_1 \cap T_2|}{2^n}$. Suppose that, from the analysis of the circuits we obtain the following superset and subset relationships: $T_3 \cup T_4 \subseteq T_1 \subseteq T_5$ and $T_6 \cup T_7 \subseteq T_2 \subseteq T_8$, where $T_1$, $T_2$, $T_3$, $T_4$, $T_5$, $T_6$, $T_7$, and $T_8$ are the test sets for faults $f_1$, $f_2$, $f_3$, $f_4$, $f_5$, $f_6$, $f_7$, and $f_8$, respectively. We can deduce the following relationship: $(T_3 \cup T_4) \cap (T_6 \cup T_7) \subseteq T_1 \cap T_2 \subseteq T_5 \cap T_8$. Hence, $1 - \frac{|T_5 \cap T_8|}{2^n} \leq d_{1,2} \leq 1 - \frac{|(T_3 \cup T_4) \cap (T_6 \cup T_7)|}{2^n}$.

This means that, if we know $T_3$, $T_4$, $T_5$, $T_6$, $T_7$ and $T_8$, then we do not need to calculate $T_1$ and $T_2$, but can obtain the above bounds on $d_{1,2}$. This can reduce the computation time for calculating the $d_{i,j}$ values of different fault pairs.

It is possible that calculation of the test sets for $f_1$, …, $f_8$ may be difficult (may take long time). However, it may be easier to calculate the $d_{i,j}$ values (using simulation techniques described in Sec. 5) for the fault pairs ($f_3$, $f_6$), ($f_4$, $f_6$), ($f_5$, $f_6$), ($f_3$, $f_7$), ($f_4$, $f_7$), ($f_5$, $f_7$), ($f_3$, $f_8$), ($f_4$, $f_8$), ($f_5$, $f_8$). Then, the $d_{1,2}$ value for the fault pair ($f_1$, $f_2$) can be approximated in the following way:

Since, $(T_3 \cup T_4) \cap (T_6 \cup T_7) \subseteq T_1 \cap T_2 \subseteq T_5 \cap T_8$, we can write $(T_3 \cap T_6) \cup (T_3 \cap T_7) \cup (T_4 \cap T_6) \cup (T_4 \cap T_7) \subseteq T_1 \cap T_2 \subseteq T_5 \cap T_8$. This implies, write $\max[|T_3 \cap T_6|, |T_3 \cap T_7|, |T_4 \cap T_6|, |T_4 \cap T_7|] \leq |T_1 \cap T_2| \leq |T_5 \cap T_8|$. Hence, $d_{5,8} \leq d_{1,2} \leq 1 - \max[(1-d_{3,6}), (1-d_{3,7}), (1-d_{4,6}), (1-d_{4,7})]$. Thus, $d_{5,8} \leq d_{1,2} \leq \min[d_{3,6}, d_{3,7}, d_{4,6}, d_{4,7}]$. Thus, we do not have to explicitly calculate the value of $d_{1,2}$ as long as we know the values of $d_{3,6}$, $d_{3,7}$, $d_{4,6}$, $d_{4,7}$.

The above illustration also holds for a multiple-output combinational logic circuit. Given a general multiple-output combinational logic circuit, we first decompose it into maximal single-output fanout-free regions and calculate the test sets and error responses for the stuck-at faults at the inputs and outputs of the fanout-free regions. It follows directly from Theorems 2 and 3 that if a test pattern $t$ detects a fault $f$ inside a single-output fanout-free region and produces an erroneous pattern $e$ at the output of the combinational logic network, then $t$ also detects the stuck-at-0 or the stuck-at-1 fault at the output of the fanout-free region and produces the same erroneous pattern $e$ at the output of the combinational logic network.

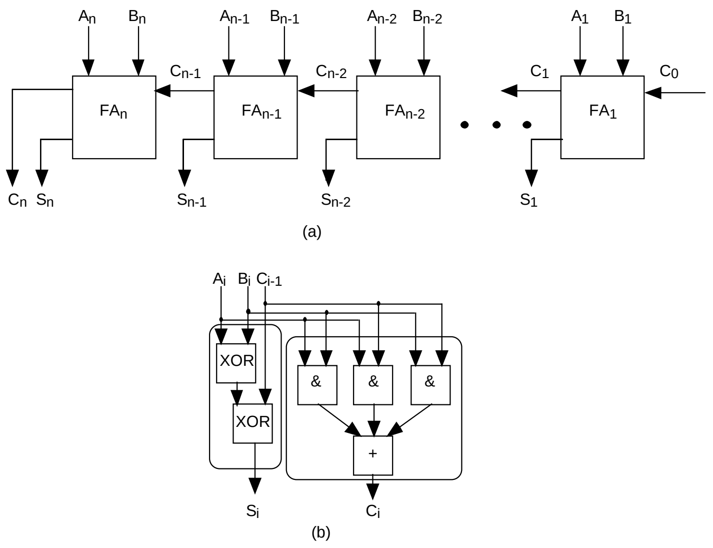

Figure 4.1. Implementation of a ripple-carry adder. (a) Ripple Carry adder using full-adder blocks. (b) Implementation of a full-adder.

Table 3.1. Reduction of the number of fault pairs

| Circuit | % Reduction |
|---|---|
| alu4 | 49 |
| apex2 | 52 |
| apex4 | 54 |
| b12 | 50 |
| cordic | 57 |
| cps | 45 |
| duke2 | 50 |
| ex1010 | 57 |
| misex2 | 41 |
| misex3 | 50 |
| pdc | 47 |
| sao2 | 57 |
| seq | 50 |
| spla | 43 |
| t481 | 80 |
| table3 | 50 |
| table5 | 45 |
| vg2 | 62 |

For estimating the potential benefits of the reduction of fault pairs, we present some simulation results on MCNC benchmark circuits. We synthesized two implementations of the truth tables with true and complemented outputs using the *Sis* tool [Sentovich 92]. For each benchmark circuit, we calculated the reduced number of fault pairs obtained by considering stuck-at faults at the inputs and outputs of fanout free sub-circuits of the two implementations. We calculate the percentage reduction in the number of fault pairs to be considered as:

$$(1-\frac{\text{Reduced number of fault pairs}}{\text{Total number of of fault pairs}})\times 100\%$$

The simulation results in Table 3.1 show that in the worst-case we obtain around 40% (i.e., 1.6 times) reduction while in the best case we obtain around 80% (i.e., 5 times reduction). While the number of fault pairs can be greatly reduced, the accuracy of the bounds can suffer. However, the user can control the extent to which the reduction must be performed depending on the desired accuracy of the bounds. For example, if the user uses only fault equivalence rules for reduction of the number of fault pairs, then the bounds will be perfectly accurate.

## 4. Diversity Calculation for Datapath Circuits

In this section we calculate the value of the D-metric for different datapath circuits like adders, priority encoders, etc. Our main focus is on datapath designs based on iterative logic networks. However, similar analysis techniques can be used for other structures (e.g., trees and combinations of iterative logic networks and trees). In this section, we illustrate the calculation technique for ripple-carry adder. Techniques for calculation of diversity for carry-select and carry-lookahead adders are described in [Mitra 00c].

Consider the design of an *n*-bit ripple-carry adder (Fig. 4.1a) using the full-adder blocks shown in Fig. 4.1b. The following theorem tells us that for a duplex system containing two identical copies of the ripple-carry adder, the $d_{1,2}$ value is 1 for any fault pair ($f_1$, $f_2$) affecting non-adjacent full-adder blocks in the two copies. This means that, for these fault pairs we do not have to explicitly calculate the value of $d_{1,2}$. This can significantly reduce the computational complexity of the D-metric.

**Theorem 4:** Consider a duplex system consisting of two identical copies $N_1$ and $N_2$ of a ripple-carry adder

(Fig. 4.1). Consider a fault $f_1$ affecting the full-adder block $FA_i$ in $N_1$ and a fault $f_2$ affecting $FA_j$ in $N_2$. If $j > i+1$ or $i > j+1$ (i.e., $f_1$ and $f_2$ affect two non-adjacent full-adder blocks in the two copies), then $d_{1,2} = 1$.

**Proof:** Consider the case when $j > i+1$. The case when $i > j+1$ is symmetric. If a fault $f_1$ in the full-adder block $FA_i$ affects $S_i$, a mismatch will be reported when $S_i$ outputs of $N_1$ and $N_2$ are compared. If fault $f_1$ affects only the $C_i$ output of $FA_i$ in $N_1$, then the $S_{i+1}$ output of $FA_{i+1}$ in $N_1$ will be erroneous. Since $|i-j| > 1$, it is guaranteed that the $S_{i+1}$ output of $FA_{i+1}$ in $N_2$ will be correct — hence, a mismatch will be reported. Hence, $d_{1,2} = 1$. Q.E.D.

For the remaining fault pairs affecting adjacent full-adder blocks (i.e., $j = i$ or $j = i+1$), we can form a circuit by cascading two full-adder blocks and calculate the exact value of $d_{i,j}$ for every fault pair. The circuit containing a cascade of two full-adder blocks has only 5 inputs and 3 outputs. Hence, the set of input combinations that produce the same erroneous output in the two copies can be calculated easily. Our results show that it takes around 7 seconds (real-time) to calculate these input combinations for all fault pairs on a SUN Ultra Sparc 2 workstation. Once these input combinations are obtained, the following procedure must be used to calculate the $d_{i,j}$ value.

Let us suppose that we are considering fault pairs in the cascade of the blocks $FA_{i+1}$ and $FA_{i+2}$ in an $n$-bit adder. For each combination of $A_{i+2}$, $B_{i+2}$, $A_{i+1}$, $B_{i+1}$ and $C_i$ for which the fault pair produces identical erroneous outputs, we calculate the number of input combinations of the full-adder that produce the particular combination of $A_{i+2}$, $B_{i+2}$, $A_{i+1}$, $B_{i+1}$ and $C_i$.

For example, suppose that for a particular fault pair, the input combination $A_{i+2} = B_{i+2} = A_{i+1} = B_{i+1} = C_i = 1$ produces identical erroneous pattern at the outputs of the cascaded block. The number of input combinations for the $n$-bit adder that satisfies the above assignment of values is $2^{2(n-i-2)} 2^{i-1} (2^{i}-1)$. Since we only have to satisfy that $A_{i+2} = B_{i+2} = A_{i+1} = B_{i+1} = 1$, we can assign any combination of 0s and 1s to the remaining bit positions $A_{i+3}, \ldots, A_n, B_{i+3}, \ldots, B_n$. This gives the term $2^{2(n-i-2)}$ in the above expression. Next, we calculate the number of input combinations, $Z_i$, of $A_1, \ldots, A_i, B_1, \ldots, B_i$, that produce $C_i$ equal to 1. The value of $Z_i$ is calculated using the following theorem proved in the Appendix A.

**Theorem 5:** The value of $Z_i$ is equal to:

- $Z_i = 2^{i-1}(2^{i}-1)$ if $c_0 = 0$ is the only value $c_0$ can have;
- $Z_i = 2^{2i}$ if $c_0$ can be both 0 or 1 with equal probability.

We can use the above techniques for other kinds of adder implementations. [Mitra 00c] shows the use of these techniques for carry look-ahead and carry-select adders. These techniques can be generalized for any iterative logic network. All these results demonstrate that for circuits exhibiting regular structures we can exploit the structural regularity to compute the value of the $D$-metric very quickly.

## 5. Diversity Estimation for General Combinational Logic Circuits

### 5.1. Signal Probability Calculation Model

In Sec. 4, we utilized the regularity in the implementation of datapath logic circuits to devise fast techniques to estimate the value of the D-metric. For general logic circuits (often called random logic circuits) we may not be able to exploit the regularity to estimate the value of the D-metric because there may not be any regularity present in the structure of general combinational logic circuits. For a given fault pair ($f_i$, $f_j$), the problem of calculating the $d_{i,j}$ value can be modeled as the *signal probability calculation* problem [Parker 75]. The modeling is shown in Fig. 5.1.

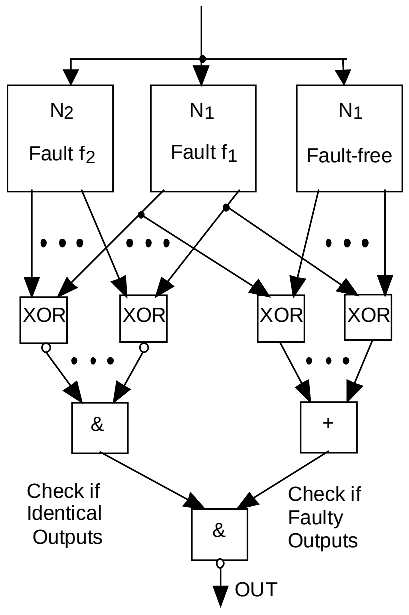

Figure 5.1. Modeling the diversity estimation problem as a signal probability calculation problem.

As shown in Fig. 5.1, we consider three blocks — $N_1$ in the presence of the fault $f_1$, $N_2$ in the presence of the fault $f_2$, and the fault-free $N_1$ block. In response to any input combination, if the incorrect outputs produced by the two faulty blocks match, then the same error pattern has been produced by the two faulty blocks. Otherwise, the two blocks either produced correct values or different error patterns at their outputs. The probability that the OUT signal is 1 is the same as the $d_{1,2}$ value for the fault pair ($f_1$, $f_2$). It is known that the signal probability calculation is a very hard problem — it is a *#P* complete problem [Motwani 97].

The Parker-McCluskey method [Parker 75] can be used to calculate the exact $d_{i,j}$ value for every fault-pair ($f_i$, $f_j$), after modeling the problem as a signal probability calculation problem as shown before. However, this method has an exponential complexity in the worst case. Methods based on Binary Decision Diagrams (BDDs) can also be used for this purpose. The cutting algorithm [Savir 90] can also be used to obtain an approximate value of the D-metric in polynomial time.

In Sec. 5.2, we describe an adaptive Monte-Carlo simulation technique to estimate the $d_{i,j}$ values of fault pairs because it is much simpler compared to other techniques.

### 5.2. Adaptive Monte-Carlo Simulation

The classical Monte-Carlo technique can be used to estimate the $d_{i,j}$ values [Motwani 97] after modeling the problem as a signal probability calculation problem. This involves choosing $N$ independent input combinations uniformly and estimating the probability that in response to a random choice, the two implementations will produce either correct values or non-identical error patterns at the outputs in the presence of the faults. The value of $N$ will be discussed later in this section. We can define random variables $Y_1, \ldots, Y_N$ as follows: $Y_k = 1$ if the two implementations produce either the correct outputs or non-identical error patterns in the presence of the faults for input combination $k$ and $Y_k = 0$ if they produce identical error patterns at their outputs for input combination $k$. The estimator $Z$ is defined to be:

$$Z = \sum_{k=1}^{N} \frac{Y_k}{N}$$

The expected value of $Z$ is given by $E(Z) = \frac{|T|}{2^n} = d_{i,j}$. Here $T$ is the set of all input combinations in response to which the two implementations produce either correct values or non-identical error patterns at their outputs. The main challenge is to determine the value of $N$ that we need to guarantee that the error in our approximation is bounded. For that purpose, we calculate the value of $N$ such that the following relationship holds:

$$\Pr[Z \geq (1+\varepsilon)d_{i,j} \text{ or } Z \leq (1-\varepsilon)d_{i,j}] \leq \delta,$$

where δ and ε are the parameters of the Monte-Carlo simulation.

Using the Chernoff bound, it has been proved in [Motwani 97] that the above relationship holds if the following bound on $N$ is satisfied: $N \geq \frac{4}{\varepsilon^2 d_{i,j}} \ln \frac{2}{\delta}$.

When the value of $d_{i,j}$ is very small, then there is a chance of having to use an exponential number of simulations to estimate the value Z within error bounds. This is the downside of the above technique. However, we can use the following approximation. As noted at the end of Sec. 2, only the high $d_{i,j}$ values significantly affect the data integrity of diverse duplex systems. Hence, for very low $d_{i,j}$ values, accuracy is not very important as long as we do not grossly over-estimate the $d_{i,j}$ value. For example, we can set a threshold that we are concerned about the $d_{i,j}$ values greater than 0.5. This means, that the number of samples $N$ we have to consider during Monte-Carlo simulation is given by $N \geq \frac{8}{\varepsilon^2} \ln \frac{2}{\delta}$.

We perform $M$ such Monte-Carlo simulations (the value of $M$ will be discussed later in this section and is derived in Appendix B) – let us suppose that the estimated $d_{i,j}$ values from these Monte-Carlo simulations are $Z_1, Z_2, \ldots, Z_M$. If one of these $Z_i$'s is less than 0.5, we conclude that the $d_{i,j}$ value is the least of all $Z_i$'s. Otherwise, the $d_{i,j}$ value is obtained by averaging the values of $Z_i$'s. We analyze this estimation technique in Appendix B and demonstrate that our Monte-Carlo simulation is adaptive and suits the current application – it provides very good estimates for high $d_{i,j}$ values and makes sure that we do not erroneously estimate very high (optimistic) values when the actual $d_{i,j}$ value is extremely small (less than 0.5). The overall algorithm is shown next.

```
Adaptive Monte-Carlo Simulation
Inputs: $M$, $N$, a duplex system, a fault pair ($f_i$, $f_j$)
Output: The $d_{i,j}$ value of a fault pair

1. Model the problem as a signal probability calculation problem (Sec. 5.1).
2. Run $M$ Monte-Carlo experiments each containing $N$ simulations.
3. Store the results obtained from each of the $M$ experiments.
4. If (the result from an experiment < 0.5) then
        The $d_{i,j}$ value of the fault pair = the minimum
        result obtained from $M$ experiments.
Else The $d_{i,j}$ value of the fault pair = average of the results
        obtained from $M$ experiments.
End
```

Next, we consider some realistic values of the various parameters ($M$, $\delta$ and $\varepsilon$) for Monte-Carlo simulation and show the reduction in the number of input combinations that must be applied to estimate the value of $d_{i,j}$. We can assume that the value of $\delta$ is $2^{-20}$.

If we can tolerate an error of 1% for high values of $d_{i,j}$ – i.e., the value of $\varepsilon$ is 0.01 – then, in each Monte-Carlo simulation we have to apply $N$ = 1,520,000 input combinations; for 2 such Monte-Carlo experiments ($M$ = 2) we have to apply a total of 3 million input combinations, and for 10 such experiments ($M$ = 10) we have to apply a total of around 15 million input combinations. When the number of inputs of the given logic function is very high (greater than 30), this technique proves to be very effective. The probability that we will erroneously make a very pessimistic estimate of $d_{i,j}$ is approximately $2^{-17}$ (= $M\delta$) (proved in Appendix B). The probability that for exponentially low values of $d_{i,j}$, we will make a very optimistic estimate greater than 0.5 is approximately $\left[\frac{e}{2^n}\right]^5$ for an $n$-input logic function (follows from Appendix B).

If we can tolerate an error of 10 % for high values of $d_{i,j}$ – i.e., the value of $\varepsilon$ is 0.1 – then, in each Monte-Carlo experiment we have to apply $N$ = 15,200 input combinations; for 10 (2) such Monte-Carlo experiments, we have to apply a total of around 150 (30) thousand input combinations. When the number of inputs of the given logic function is very high (greater than 20), this technique proves to be very effective.

Table 5.1 illustrates the advantage of using adaptive Monte-Carlo simulation technique described in this section for various values of percentage errors ($\varepsilon$) compared to exhaustive simulation.

Table 5.2 shows some simulation results that demonstrate the accuracy of the adaptive Monte-Carlo simulation technique. Since the problem of calculating the $d_{i,j}$ values can be modeled as the signal probability calculation problem (Sec. 5.1), we used the adaptive Monte-Carlo technique for estimating the signal probabilities of the logic functions implemented by some MCNC benchmark circuits and compared the results with those obtained by exhaustive simulation. The benchmark circuits considered are such that exhaustive simulation is feasible. For the adaptive Monte-Carlo simulation technique, we used different parameters; (1) 1% error and 2 Monte-Carlo experiments with 1.52 million simulations in each experiment; (2) 1% error and 10 Monte-Carlo simulation experiments with 1.52 million simulations in each experiment; (3) 10% error and 2 Monte-Carlo simulation experiments with 0.0152 million simulations in each experiment; and, (4) 10% error and 10 Monte-Carlo experiments with 0.0152 million simulations in each experiment. The results in Table 5.2 show that the theoretical arguments behind adaptive Monte-Carlo simulation are applicable for real-life circuits and demonstrate the effectiveness of the adaptive Monte-Carlo simulation technique.

## 6. Conclusions

This paper demonstrates the feasibility of calculating the value of the design diversity metric for arbitrary combinational logic circuits. Although the general problem is NP-complete, efficient algorithms can be devised for solving the problem. For datapath logic circuits and circuits with iterative logic networks, the regularity in the circuit structures can be exploited to compute the value of the diversity metric very fast. For general combinational logic circuits, reduction techniques using fault equivalence and fault dominance relationships can be applied to significantly reduce the number of fault pairs to be considered during diversity calculation. Next, the adaptive Monte-Carlo simulation technique can be used to obtain accurate estimates of the diversity metric for the reduced set of fault pairs using the number of simulations which is polynomial (instead of exponential) in the number of inputs of the combinational logic function. Moreover, the number of simulations to be used can be tuned depending on the error that can be tolerated during estimation. This paper describes techniques for estimating diversity in combinational logic circuits; a related paper [Mitra 00b] describes design diversity estimation techniques for sequential logic circuits.

Table 5.1. Advantages of adaptive Monte-Carlo simulation technique over exhaustive simulation by comparing the number of input combinations that must be simulated

| # circuit inputs | 2 Monte-Carlo experiments | | 10 Monte-Carlo experiments | | Exhaustive Simulation |
|---|---|---|---|---|---|
| | $\varepsilon$ = 0.1 | $\varepsilon$ = 0.01 | $\varepsilon$ = 0.1 | $\varepsilon$ = 0.01 | |
| 20 | 0.03 Million | 1.04 Million | 0.152 Million | 1.04 Million | 1.04 Million |
| 25 | 0.03 Million | 3 Million | 0.152 Million | 15 Million | 33.5 Million |
| 30 | 0.03 Million | 3 Million | 0.152 Million | 15 Million | 1074 Million |
| 35 | 0.03 Million | 3 Million | 0.152 Million | 15 Million | 34359 Million |
| 40 | 0.03 Million | 3 Million | 0.152 Million | 15 Million | – |
| 45 | 0.03 Million | 3 Million | 0.152 Million | 15 Million | – |

Table 5.2. Adaptive Monte-Carlo simulation results for some MCNC benchmark circuits

| Circuit name | # circuit inputs | 2 Monte-Carlo experiments | | 10 Monte-Carlo experiments | | Exhaustive Simulation |
|---|---|---|---|---|---|---|
| | | $\varepsilon = 0.1$ | $\varepsilon = 0.01$ | $\varepsilon = 0.1$ | $\varepsilon = 0.01$ | |
| cordic | 23 | 0.928 | 0.93 | 0.929 | 0.93 | 0.934 |
| | | 0.1001 | 0.099 | 0.1001 | 0.0988 | 0.11 |
| | | 0.372 | 0.374 | 0.37 | 0.374 | 0.376 |
| misex2 | 25 | 0.00019 | 0.0002 | 0.00019 | 0.0002 | 0.0002 |
| | | 0.246 | 0.2501 | 0.246 | 0.249 | 0.25 |
| | | 0.487 | 0.486 | 0.484 | 0.486 | 0.471 |
| vg2 | 25 | 0.481 | 0.483 | 0.48 | 0.483 | 0.468 |
| | | 0.00006 | 0.00005 | 0 | 0.00004 | 0.000041 |

## 7. Acknowledgment

This work was supported by Defense Advanced Research Project Agency (DARPA) under Contract No. DABT63-97-C-0024 (ROAR project). Thanks to Mr. Mayur Datar of the Department of Computer Science, Stanford University, for helpful discussions.

## 8. References

[Avizienis 84] Avizienis, A. and J. P. J. Kelly, “Fault Tolerance by Design Diversity: Concepts and Experiments,” *IEEE Computer*, pp. 67-80, August 1984.

[Briere 93] Briere, D. and P. Traverse, “Airbus A320/A330/A340 Electrical Flight Controls: A family of fault-tolerant systems,” *Proc. FTCS*, pp. 616-623, 1993.

[Garey 79] Garey, M. and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[Lala 94] Lala, J. H. and R. E. Harper, “Architectural principles for safety-critical real-time applications,” *Proc. of the IEEE*, vol. 82, no. 1, pp. 25-40, January, 1994.

[Lyu 91] Lyu, M. R. and A. Avizienis, “Assuring design diversity in N-version software: a design paradigm for N-version programming,” *Proc. DCCA*, pp. 197-218, 1991.

[McCluskey 88] McCluskey, E. J., S. Makar, S. Mourad and K. D. Wagner, “Probability Models for Pseudo-random Test Sequences,” *IEEE Trans. Computers*, Vol. 37, No. 2, pp. 160-174, Feb. 1988.

[McCluskey 00] McCluskey, E. J., and C. W. Tseng, “Stuck-at Faults vs. Actual Defects,” *Proc. Intl. Test Conf.*, pp. 336-343, 2000.

[Mitra 99a] Mitra, S., N. R. Saxena and E. J. McCluskey, “A Design Diversity Metric and Reliability Analysis of Redundant Systems,” *Proc. IEEE Intl. Test Conf.*, pp. 662-671, 1999.

[Mitra 99b] Mitra, S., N. R. Saxena and E. J. McCluskey, “A Design Diversity Metric and Analysis of Redundant Systems,” *Technical Report*, *Center for Reliable Computing*, *Stanford Univ.*, CRC-TR-99-4, 1999.

[Mitra 00a] Mitra, S., N. R. Saxena and E. J. McCluskey, “Common-Mode Failures in Redundant VLSI Systems: A Survey,” *IEEE Trans. Reliability*, Special Section on Fault-Tolerant VLSI Systems, 2000, To appear.

[Mitra 00b] Mitra, S., and E. J. McCluskey, “Design Diversity for Concurrent Error Detection in Sequential Logic Circuits,” *VLSI Test Symp.*, 2001.

[Mitra 00c] Mitra, S., N. R. Saxena and E. J. McCluskey, “Techniques for Estimation of Design Diversity for Combinational Logic Circuits,” *Technical Report, Center for Reliable Computing, Stanford University, CRC-TR-01-1*, 2001 (http://crc.stanford.edu).

[Motwani 97] Motwani, R., and P. Raghavan, *Randomized Algorithms*, 1997.

[Parker 75] Parker, K.P., and E.J. McCluskey “Probabilistic Treatment of General Combinational Networks,” *IEEE Trans. Computers,* Vol. C-24, No. 6, pp. 668-670, June 1975.

[Pradhan 96] Pradhan, D. K., *Fault-Tolerant Computer System Design,* Prentice Hall, 1996.

[Rabaey 96] Rabaey, J., *Digital Integrated Circuits*, Prentice Hall, Englewood Cliffs, 1996.

[Riter 95] Riter, R., “Modeling and Testing a Critical Fault-Tolerant Multi-Process System,” *Proc. FTCS*, pp. 516-521, 1995.

[Savir 90] Savir, J., “Improved Cutting Algorithm,” *IBM Journal Res. and Dev.*, Vol. 34, No. 2-3, pp. 381-388, March-May 1990.

[Siewiorek 92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 1992.

[Siewiorek 92] Siewiorek, D. P. and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, Digital Press, 1992.

[Tamir 84] Tamir, Y. and C. H. Sequin, "Reducing common mode failures in duplicate modules," *Proc. ICCD,* pp. 302-307, 1984.

[To 73] To, K., "Fault Folding for Irredundant and Redundant Combinational Circuits," *IEEE Trans. Computers*, Vol. C-22, No. 11, pp. 1008-1015, Nov. 1973.

[Tohma 71] Tohma, Y. and S. Aoyagi, "Failure-tolerant sequential machines with past information," *IEEE*

*Trans. Computers*, Vol. C-20, No. 4, pp. 392-396, April 1971.

## Appendix A

**Theorem 5:** The value of $Z_i$ is equal to:

- $Z_i = 2^{i-1}(2^i-1)$ if $c_0 = 0$ is the only value $c_0$ can have;
- $Z_i = 2^{2i}$ if $c_0$ can be both 0 or 1 with equal probability.

**Proof:** The value of $x_i$ can be expressed using the following recurrence relation:

$$Z_i = 2^{2(i-1)} + 2Z_{i-1}, \text{ and}$$

$Z_1 = 1$ if $c_0 = 0$, and $Z_1 = 4$ if $c_0 = 0$ and $c_0 = 1$ are equally likely.

The above recurrence relation follows from the fact that when $A_i = 1$ and $B_i = 1$, then the value of $C_i$ will be 1, irrespective of the remaining bits. However, when $A_i = 0$ and $B_i = 1$, or vice-versa, then we want $C_{i-1}$ to be equal to 1. When $i$ =1, then only $A_i = 1$ and $B_i = 1$ can make $C_i$ equal to 1. The solution to the above recurrence relation is given by $Z_i = 2^{i-1}(2^i-1)$. This can be obtained very easily by using the simple technique of generating functions.

If $c_0 = 0$ and $c_0 = 1$ are equally likely, $Z_1 = 4$ and hence, $Z_i = 2^{2i}$ from the above recurrence relation.

Q.E.D.

## Appendix B

There can be two kinds of errors in the adaptive Monte-Carlo estimation technique. The actual $d_{i,j}$ value can be high but we can erroneously declare that the estimated $d_{i,j}$ value is very low. Although this situation will produce pessimistic values, it is not desired. As noted in Sec. 5.2, δ is the probability that the value of $d_{i,j}$ is out of the error bound. Hence, the probability that the $d_{i,j}$ value is erroneously declared to be less than 0.5 is less than $1-[1-\delta]^M \approx M\delta$ for $\delta << 1$.

Thus, if we choose $M$, δ and ε appropriately, then we can obtain close approximation of the $d_{i,j}$ values.

The other source of error is due to the fact that the actual $d_{i,j}$ value may be less than 0.5 but for all the Monte-Carlo simulation experiments the value is estimated to be greater than or equal to 0.5. The worst scenario is when the actual $d_{i,j}$ value is exponentially small but the estimated value is close to 1. We show next that the probability of such an event is extremely small and is almost negligible. The following fact has been proved in [Motwani 97] (using Chernoff bound): $\Pr[Z > (1+\alpha)\mu] < \left[\frac{e^{\alpha}}{(1+\alpha)^{(1+\alpha)}}\right]^{\mu}$, where $\mu = \frac{|T|}{2^n}$. If $\mu = \frac{0.5}{x}$ where $x$ is a real number, $\alpha$ must be equal to $x$-1 for $(1 + \alpha)\mu$ to be equal to 0.5. This means, $\Pr[Z > 0.5] < \frac{e^{0.5}}{x^{0.5}}$. The probability that the value of $Z$ will be greater than 0.5 in all the $M$ experiments is less than $\left[\frac{e}{x}\right]^{\frac{M}{2}}$. When the value of $\mu$ is exponentially small, $x$ is of the order of $2^n$, and this probability becomes extremely small.

Thus, our Monte-Carlo simulation is adaptive and suits the current application – it provides very good estimates for high $d_{i,j}$ values and makes sure that we do not erroneously estimate very high (optimistic) values when the actual $d_{i,j}$ value is extremely small (less than 0.5).