Center for Reliable Computing

TECHNICAL REPORT

Design of Redundant Systems Protected Against Common-Mode Failures

Subhasish Mitra and Edward J. McCluskey

00-3

(CSL TR # ??)

March 2000

Center for Reliable Computing
Gates Building 2A, Room 236
Computer Systems Laboratory
Dept. of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305-9020

Abstract:

Redundancy techniques like duplication and Triple Modular Redundancy (TMR) are widely used to design fault-tolerant systems. In this paper, for the first time, we develop fault models for common-mode failures (CMFs) in redundant systems and describe techniques to design redundant systems protected against the modeled CMFs. We first develop an input-register-CMF model that targets systems with register-files. This paper shows that, in the presence of input-register-CMFs, we can always design duplex or TMR systems that either produce correct outputs or indicate error situations when incorrect outputs are produced. This property ensures data-integrity and is called the fault-secure property. Next, we extend the input-register-CMF model to consider systems where the storage elements of the registers are not organized in register-files; instead, the register flip-flops are placed using conventional CAD programs. For this case, we modify our previous technique to synthesize redundant systems that are fault-secure against the extended input-register-CMFs. Our results show that redundant systems obtained using our technique require comparable or less area than conventional redundant systems that are not protected against CMFs.

Funding:

This work was supported by the Advanced Research Projects Agency under prime contract No. DABT-97-C-0024.

Imprimatur: Philip Shirvani and Nahmsuk Oh

DESIGN OF REDUNDANT SYSTEMS PROTECTED AGAINST COMMON-MODE FAILURES

(CRC-TR-00-2 Preliminary Version)

Subhasish Mitra and Edward J. McCluskey
Center For Reliable Computing
Departments of Electrical Engineering and Computer Science
Stanford University
Gates Building 2A, Room 236, MC9020
Stanford, CA 94305
Voice: (650)723-1258 FAX: (650)723-1451
Email: {smitra, ejm}@crc.Stanford.EDU

ABSTRACT

Redundancy techniques like duplication and Triple Modular Redundancy (TMR) are widely used to design fault-tolerant systems. In this paper, for the first time, we develop fault models for common-mode failures (CMFs) in redundant systems and describe techniques to design redundant systems protected against the modeled CMFs. We first develop an input-register-CMF model that targets systems with register-files. This paper shows that, in the presence of input-register-CMFs, we can always design duplex or TMR systems that either produce correct outputs or indicate error situations when incorrect outputs are produced. This property ensures data-integrity and is called the fault-secure property. Next, we extend the input-register-CMF model to consider systems where the storage elements of the registers are not organized in register-files; instead, the register flip-flops are placed using conventional CAD programs. For this case, we modify our previous technique to synthesize redundant systems that are fault-secure against the extended input-register-CMFs. Our results show that redundant systems obtained using our technique require comparable or less area than conventional redundant systems that are not protected against CMFs.

Keywords: Redundancy, Common-mode failures, On-line Testing, Fault Model, Data Integrity, Synthesis

1. INTRODUCTION

Redundancy techniques like duplication and Triple Modular Redundancy (TMR) are often used for designing fault-tolerant systems. Triple-Modular-Redundancy (TMR) [Von Neumann 56] is an example of a hardware redundancy scheme that enables us to achieve high reliability through fault-masking. In a TMR system, we replicate three copies of a particular module and the outputs of all the modules are connected to a voter. Such a technique produces correct results as long as two modules function correctly (i.e., only one of the three modules can fail). There is a vast literature on different hardware redundancy techniques and reliability modeling for systems incorporating these redundancy techniques [Trivedi 82].

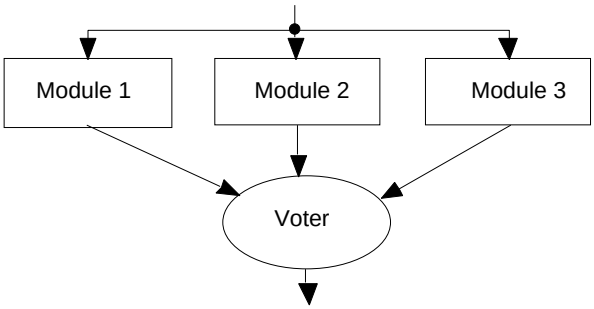

Figure 1.1. Classical triple-modular redundancy

The classical reliability expressions for redundant systems are optimistic because they do not consider common-mode failures. Lala observed that we must pay attention to the problem of common-mode failures (CMFs) [Lala 93]. CMFs result from failures that affect more than one module of the redundant system at the same time, generally due to a common cause. They may be design faults or operational faults; may be externally caused (such as EMI and radiation) or internal. However, in the literature, there is no fault-model available for CMFs although it has been pointed out that analysis techniques that include CMFs have very little causal relationship with the actual causes of CMFs. This impedes progress in the analysis and design of redundant systems in the presence of common-mode failures.

Our main contributions in this paper are the following. For the first time, we have developed real quantitative fault models for common-mode failures. We have also developed techniques to protect redundant systems against the modeled CMFs.

Section 2 introduces TMR systems with modified voter design compared to bit-wise voters used in conventional TMR systems. These voters are useful especially in the context of CMFs. In Sec. 3 we develop a common-mode fault model called input-

register-CMF for redundant systems containing register-based designs. This fault model mainly targets systems where the registers are organized in register-files. Section 4 uses an example to explain our technique to design TMR systems protected against modeled CMFs. We formulate our technique as a Boolean Satisfiability Problem in Sec. 5. In Sec. 6 we design TMR systems providing guarantee that, in the presence of modeled CMFs, they produce correct outputs or indicate erroneous situations when incorrect outputs are produced. This property ensures that data integrity is maintained in the systems and is called the *fault-secure* property in the fault-tolerance literature. Section 7 extends the input-register-CMF model to consider systems where the register flip-flops are not organized in register-files and modifies our earlier technique to design redundant systems protected against the modeled CMFs. In Sec. 8, we present simulation results. Finally, we provide summary and conclusions in Sec. 9.

2. TMR SYSTEMS WITH MULTIPLE OUTPUTS

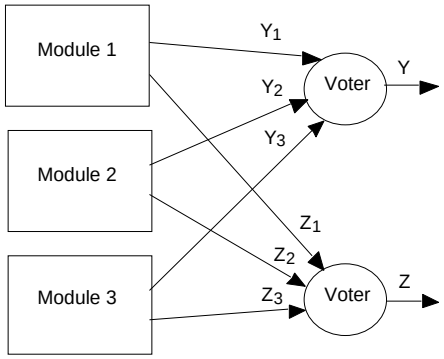

Figure 2.1. A conventional TMR system for multi-output circuits

In TMR systems, majority voting is normally performed on a bit-by-bit basis [Siewiorek 92]. For a module with *n* output lines, the TMR implementation has three modules and *n* single-bit voters. Figure 2.1 shows the implementation of such a TMR system with two outputs.

Table 2.1. Illustration of a TMR system with bit-by-voting

Module	Fault-free Outputs	Faulty Outputs
1	0 1	**1 0**
2	0 1	**1** 1
3	0 1	0 1

Consider the following example. Suppose that we have a TMR system where each module has two outputs. Due to the presence of a fault in Module 1, in response to a particular input combination, the module produces an output combination 10 instead of 01. Similarly, due to the presence of a fault in Module 2, the output combination obtained from it is 11. Finally, suppose that Module 3 is working correctly and produces

the expected output 01. This is shown in Table 2.1. With bit-wise voting, the voter corresponding to the first output produces a 1 and the one corresponding to the second output produces a 1. Thus, we have 11 at the system output. However, if we consider the *output vector* from each module, we find that the output vectors from all the three modules are different. The output vectors from the first, second and third modules are 10, 11 and 01, respectively. This can be treated as an erroneous state for a voter (which works on a majority voting principle) because, no two output *vectors* are equal. On the basis of this observation, we can modify the classical voter design by adding some extra circuitry that detects this error condition and produces an error signal. The details of the voter design can be obtained from [Mitra 00]. For the rest of the paper, we will use TMR systems with the modified voter as shown in Fig. 2.2.

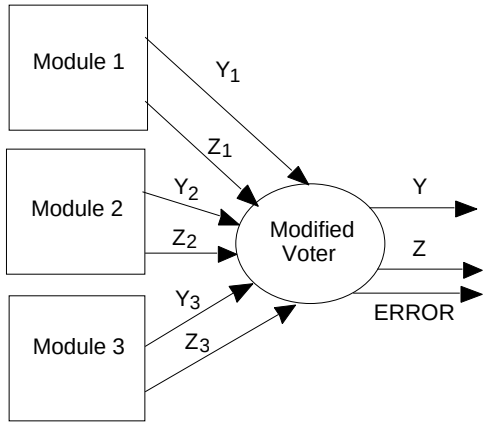

Figure 2.3. A TMR system with the modified voter design of Fig. 2.2.

3. INPUT-REGISTER-CMF MODEL

As mentioned in Sec. 1, a common-mode failure (CMF) affects more than one module of a redundant system at the same time, generally due to a common-cause. Consider a redundant system (duplex or TMR) where there are separate input registers associated with each module. In such a system, let us consider the class of failures, in the presence of which, the same bit positions in two or three of these input registers get *stuck* at the same value. In classical redundant systems, these failures will have identical effects on all the implementations and will go undetected producing incorrect outputs. Thus, these failures are possible CMF candidates. We call these CMFs *Input-Register-Common-Mode-Failures* and the corresponding fault model as the *Input-Register-CMF Model* (IR-CMF). This fault model includes transient faults from radiation upsets and permanent faults that result from permanent system interference caused by the operational environment. In a space environment, radiation damage can be a source of

common-mode faults. It was shown in [Reed 97] that radiation sources cause multiple-event upsets in sequential elements (e.g. flip-flops) of a design. Experimental results in [Liden 94] show that, in a radiation environment, 98% of the bit errors in sequential elements are caused by particles directly hitting these elements, and only 2% are caused by transients in combinational logic. It has been shown in the literature that upsets from radiation sources can be modeled as bit-flips or bit-stuck-at faults [Choi 93][Rimen 94].

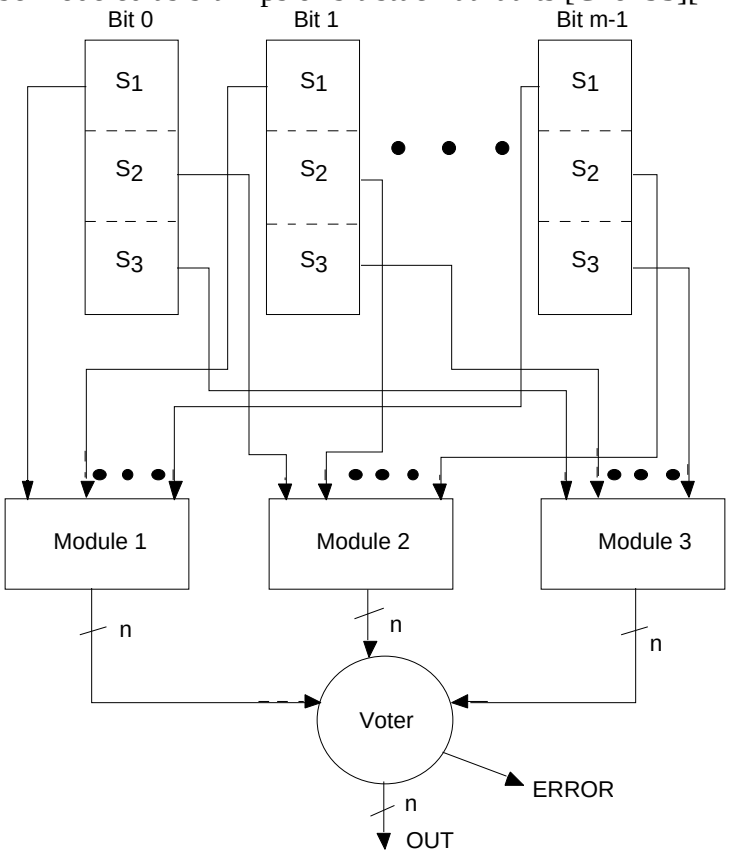

Figure 3.1. A system architecture with register files

Consider the TMR system shown in Fig. 3.1. The system architecture is such that where the registers are organized in a register file like the organization of RAM cells. The three input registers under consideration are S_1, S_2 and S_3. The storage elements in a register file are organized in an array structure. This architecture of register organization is common in many VLSI designs including microprocessors and ASICs like network switches and graphics chips. The register-file in Fig. 3.1 has three readout ports.

Let us suppose that the storage elements in a row correspond to the different bit positions of a register. In such a system, a single source of radiation can cause upsets in adjacent storage elements. If the affected storage elements belong to a particular row, then the TMR system is protected (since only one of the modules has corrupt inputs). However, if the adjacent storage elements belong to a particular column, then identical

bit positions of two or more input registers are affected by the radiation source. These CMFs cannot be detected in conventional TMR systems even using our modified voter design. It may be argued that if the registers producing inputs for the TMR system are physically placed far enough in the register-file, then the chances of upsets in two registers of the TMR system are low. However, it may not be possible to control the actual choice of the registers if compilers do not have the knowledge about the relative physical locations of the different registers.

It has been noted in the literature that design faults constitute a dominant source of CMFs in redundant systems [Lala 94]. As an example of a design fault, suppose that due to design faults, a bit-column of a register file produces very weak signals corresponding to a particular logic value. Such a design fault can manifest itself as a stuck-at fault in the same bit positions of multiple input registers.

Note that, while the focus of this paper is on bit-stuck-at-faults, our techniques can be extended for bit-flips. In the next section, with the help of an example, we describe our technique to detect the input-register-CMFs. For duplex systems, our technique stores the actual input values in the input register of one of the modules and the complemented input values in the input register of the other module. For TMR systems, our technique stores the actual input values in the input register of the first module, the complemented input values in the input register of the second module and a *transformation* of the input values in the input register of the third module.

3. AN EXAMPLE

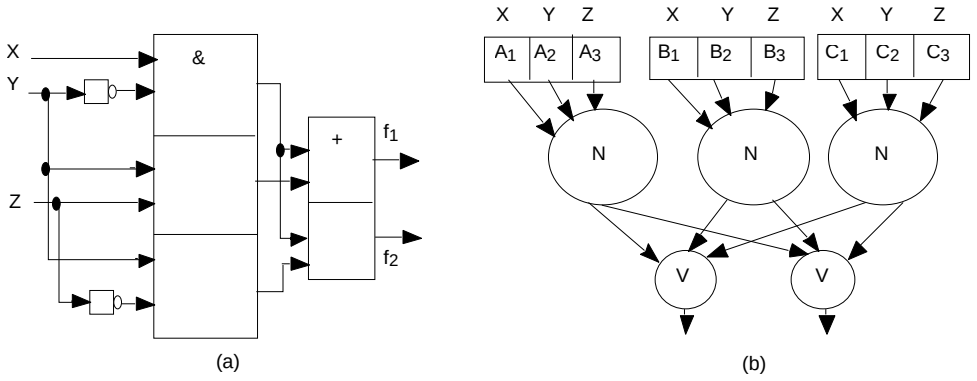

Figure 4.1. Conventional TMR (a) A multi-output circuit (b) TMR implementation

Let us consider a simple combinational logic circuit, N, with three inputs *x, y* and *z* and two outputs. The two output functions are $f_1 = xy' + yz$ and $f_2 = xy' + yz'$. The logic diagram is shown in Fig. 4.1(a). For duplex systems, we can make the system fault-secure against input-register-CMFs by storing the actual input values in the register of

one of the modules and storing the complemented input values in the input register of the other module. Thus, for input-register-CMFs, design of duplex systems is simple; the design of TMR systems is more challenging.

Figure 4.1(b) shows a TMR system for the logic function of Fig. 4.1a. We have three 3-bit registers A, B and C. We have three copies of the original circuit N. The first copy of N gets its inputs from the 3 bits of register A; i.e., A_1 stores values corresponding to X, A_2 stores Y values and A_3 stores Z values. The same scenario holds for registers B and C. The common-mode faults involving the first bit position are $\{A_1/1, B_1/1\}$, $\{A_1/1, C_1/1\}$, $\{A_1/1, B_1/1, C_1/1\}$, $\{B_1/1, C_1/1\}$, $\{A_1/0, B_1/0\}$, $\{A_1/0, B_1/0, C_1/0\}$, etc. These common-mode faults that affect a single bit position of two or more input registers are called single-bit input-register CMFs.

Table 4.1 Truth tables (a) Network N (b) Network N_1 (c) Network N_2

(a)

A_1 A_2 A_3	f_1 f_2
0 0 0	0 0
0 0 1	0 0
0 1 0	0 1
0 1 1	1 0
1 0 0	1 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 0

(b)

B_1 B_2 B_3	f_1 f_2
0 0 0	1 0
0 0 1	0 1
0 1 0	1 1
0 1 1	1 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 0
1 1 1	0 0

(c)

C_1 C_2 C_3	f_1 f_2
0 0 0	0 0
0 0 1	1 1
0 1 0	0 0
0 1 1	1 1
1 0 0	0 1
1 0 1	0 1
1 1 0	1 0
1 1 1	1 0

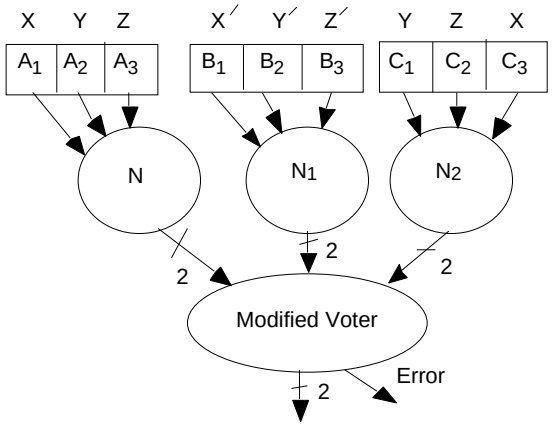

Figure 4.2. TMR implementation of the circuit of Fig. 4(a) using our new technique

Consider the TMR system of Fig. 4.2. Note that, the three registers A, B and C contain different values. The first bit of register A stores the value of variable X, the second bit stores Y's and the third bit stores Z's. The first, second and third bits of register B store values corresponding to X′, Y′ and Z′, respectively. Network N_1 may be implemented as a *dual* of the original network N [McCluskey 86], instead of replicating N and adding inverters at the inputs. *This can possibly help in protecting the system*

against common-mode failures affecting the leads inside the implemented networks, instead of only the network inputs. For register C, the first bit stores Y's, the second bit stores Z's and the third bit stores X's.

The networks are implemented in such a way that for a particular combination of X, Y and Z, all the three networks produce the same outputs. When X = 1, Y = 0 and Z = 0, all the three networks produce 11. The system architecture may be such that, for example, the second module will have complemented inputs and will produce complemented outputs. In that case, the voter design can be changed to incorporate this architectural requirement. Note that the input register of the third module contains a rotated version of the content of the input register of the first module. Hence, network N_2 can be implemented with the same number of logic gates as network N. The truth tables of N, N_1 and N_2 are shown in Table 4.1(a), 4.1(b) and 4.1(c), respectively.

We analyze the behavior of the TMR system of Fig. 4.2 in the presence of common-mode faults. For example, consider the common-mode fault $\{A_1/1, B_1/1, C_1/1\}$. If we apply X = 0, Y = 0 and Z = 1, the network N sees 101 (since A_1 is stuck at 1) at its inputs and produces $f_1 = 1$ and $f_2 = 1$. The network N_1 sees 110 at its inputs and produces 00 at its output. Network N_2 sees 110 at its input (because C_1 is stuck at 1 and X = 0, Y = 0 and Z = 1) and produces 10 at its output. Thus, when X = 0, Y = 0 and Z = 1, the three modules produce three different vectors at their outputs. This is an erroneous situation for the voter (as explained in Sec. 2) and hence, the *ERROR* signal is turned on by the voter. Hence, we can detect the common-mode fault under consideration. In a similar way, it can be shown that other common-mode faults can be detected by the TMR system in Fig. 4.2. The system is fault-secure against the modeled IR-CMFs. It may be argued whether $\{A_1/1, C_3/1\}$ should also be called a common-mode fault. For the initial part of this paper, we are *not* going to consider it as an IR-CMF. However, later in this paper, we will present a technique to handle these cases too. Thus, our goal is to find suitable transformations for arbitrary functions, such that TMR systems are protected against input-register-CMFs. In the next section, we will describe a technique to solve this problem.

5. A TECHNIQUE BASED ON BOOLEAN SATISFIABILITY

As explained in the previous section, we want to find a transformation for the contents of the input register of the third module of a TMR system in order to protect it against IR-CMFs. In this section we describe a technique to achieve this goal. For the ease of explanation, we will describe our technique to design TMR systems that are self-testing against input-register-CMFs. The *self-testing* property ensures that for every common-mode fault, there exists at least one input combination of X, Y and Z for which

the fault will be detected (the error signal will be produced by the voter). We can easily extend the technique to consider the fault-secure property.

We assume that we are given the truth table for an *n* input logic circuit with *m* (*m* > 1) output functions. Let us suppose that the *n* inputs are $x_1, x_2, ..., x_n$. As explained in Sec. 4, we assume that in the TMR system, there will be one module N implementing the actual logic function with inputs $x_1, ..., x_n$. There will be another module N_1 with inputs $x_1', x_2', ..., x_n'$. The truth table for N_1 can be directly obtained from that of N in the following way. If an input combination of $x_1, ..., x_n$ in N produces an output combination $y_1, ..., y_m$, then an input combination $x_1', x_2', ..., x_n'$ produces the output combination $y_1, ..., y_m$ in N_1. This is because, the inputs of module N_1 are obtained by inverting the inputs of module N. With this choice of N and N_1, we can make sure that any single IR-CMF involving the input registers of N and N_1, will produce incorrect outputs for N or N_1, but not for both at the same time.

For the third module N_2, we want to find a transformation function T such that, for each input-register-CMF, there exists at least one input combination that will result in turning on the *ERROR* output of the modified voter. For the present discussion, we will assume that T is a *bijection*. This means that for every input combination for the module N, there exists one and only one input combination for the module N_2 in the TMR system. This problem can be formulated as a Boolean Satisfiability problem [Gary 79]. Note that, it is not necessary that T should always be a bijection. We will discuss this further in Sec. 7. In this section, we explain the types of constraints generated with the help of an example. The actual Boolean Satisfiability model is reported in Appendix A.1. Once we generate the constraints, standard Boolean Satisfiability Solvers [Ashar 91][Zhang 93] can be used to solve the Boolean Satisfiability problem.

Table 5.1 An example truth table

x y z	f_1 f_2
0 0 0	0 0
0 0 1	0 0
0 1 0	0 1
0 1 1	1 0
1 0 0	1 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 0

Table 5.2 An example transformation T

Input	T(Input)
a = 0 0 0	0 0 0
b = 0 0 1	0 1 0
c = 0 1 0	1 0 0
d = 0 1 1	1 1 0
e = 1 0 0	0 0 1
f = 1 0 1	0 1 1
g = 1 1 0	1 0 1
h = 1 1 1	1 1 1

Consider the example in Table 5.1. Let us consider an IR-CMF due to which, the first bit (bit 1) of all the input registers are stuck at 1. Let us name the eight input combinations in the following way: *a* = 000, *b* = 001, *c* = 010, *d* = 011, *e* = 100, *f* = 101, *g* = 110 and *h* = 111. Consider the input combination *a* (000). Since the first bit of all the input registers is stuck at 1, network N has 100 at its inputs and produces output

vector 11 instead of 00. The second module produces 00. In the fault-free case, the third module has T(*a*) at its inputs. Now, *c* = 010 is an input combination that produces an output vector 01 that is different from 00 or 11. If T(*a*) + 100 = T(*c*), then the third module will produce 01 and this common-mode fault will be detected by the voter (i.e., the ERROR signal will be turned on). Here, "+" means a bitwise-OR operation. This constraint says, that the encoding of *a* and *c* must differ *only* in the first bit and the encoding of *a* should have a 0 on the first bit and that of *c* should have a 1. Similarly, other constraints that are generated are: T(*a*) + 100 = T(*d*), T(*a*) + 100 = T(*g*) and T(*a*) + 100 = T(*h*). At least one of the above constraints must be satisfied in order to detect the CMF. Consider the transformation function T shown in Table 5.2.

It is obvious, that the T transformation is a bijection. Since T(000) = 000 and T(010) = 100, T(*a*) + 100 = T(*c*) is satisfied. Thus, we make sure that if the first bit of the input registers of module 1, module 2 and module 3 are stuck-at 1 due to the presence of a CMF, an error will be reported by the system. It can be shown that with the transformation in Table 5.2, all other input-register-CMFs can be detected.

6. LOGIC FUNCTION AND TRANSFORMATION CHARACTERIZATION

In this section, we first characterize the class of functions for which the technique of Sec. 5 can be applied to find transformations for the inputs to the third module, so that all single IR-CMFs are detected. We present a set of theorems (without proofs for brevity).

Theorem 1: For any multi-output logic function with *n* inputs, producing at least *n*+5 distinct output combinations, we can always find a transformation T such that every single input-register-CMF can be detected.

Table 6.1. An example to illustrate the bound in Theorem 2

a b c d	f_1 f_2 f_3 f_4
0 0 0 0	0 1 1 0
0 0 0 1	0 1 0 0
0 0 1 0	1 0 0 1
0 0 1 1	0 1 1 1
1 1 0 0	0 1 0 1
1 1 0 1	1 0 0 0
1 1 1 0	1 1 0 1
1 0 1 0	1 0 1 0
others	0 0 0 0

For example, consider a logic function with 4 input variables and 9 output vectors, as shown in Table 6.1. Thus, the bound in Theorem 1 is satisfied.

Table 6.2. The MCNC benchmark circuits that satisfy the bound in Theorem 1

Circuit Name	# Inputs	# Outputs
Z5xp1	7	10
apex1	45	45
apex3	54	50
apex4	9	19
cps	24	109
ex1010	10	10
ex5	8	63
inc	7	9
misex3c	14	14
pdc	16	40
spla	16	46
squar5	5	8
table3	14	14
table5	17	15

We considered the combinational logic functions from the MCNC benchmark suite to find which of them satisfy the bound given in Theorem 1. For the circuits with *don't cares* in the outputs, we fixed the don't cares to 0s for the ease of experimentation. Table 6.2 shows the circuits that satisfy the bound in Theorem 1. Some MCNC logic functions (around 10) did not satisfy the bound in Theorem 1 because of the following reasons. We fixed all the don't cares in the outputs to 0's. Also, some of these logic functions produce mainly one-hot outputs (all output combinations have no 1 or only one 1 and rest 0s).

This shows that for designing TMR systems for these logic functions, we can use our technique in order to make the system is self-testing against input-register-CMFs. It can be proved Theorem 1 also holds for datapath circuits like adders, multipliers, etc.

In Sec. 5 we presented a formulation of the problem of finding a suitable transformation T for the inputs of the third module in a TMR system such that all single input-register-CMFs are detected. However, for practical purposes, proving the existence of such a transformation function is not enough. It is obvious that the logic complexity of the third module is dependent on the transformation chosen. Thus, depending on the function being implemented, we want to choose a *good* transformation function such that the complexity of the logic implementing the third module does not blow up. This means that, for the Boolean Satisfiability Problem used to model the problem of finding a transformation T, we have to consider all possible satisfying transformations and choose the one that leads to the minimum logic implementation. However, there is an inherent difficulty in solving such a problem for circuits with a large number of inputs. This is because, the time complexity of the Satisfiability problem is exponential in the number of variables. Thus, it is important to consider some restricted transformations and examine the class of functions for which these transformations can be applied.

First, we characterize the left rotation transformation. By a *left rotation transformation*, we mean that the content of the input register of the third module is obtained by a left rotation of that of the input register of the first module. For example, suppose that the input register of the first module contains values of the variables *x, y* and *z* in the first, second and third bit positions, respectively. If the input register of the third module contains values corresponding to y, z and x in its first, second and third bit positions, respectively, then we have a left rotation transformation. Note that, this transformation does not have any effect on the logic complexity of the network corresponding to the third module. Next, we show that by adding at most two extra output bits, any TMR system can be made fault-secure against input-register-CMFs if we use this transformation.

Theorem 2: Any function with an *even* number (*n*) of input bits can be made fault-secure with respect to the input-register-CMFs, by adding a maximum of one extra output bit, if we apply the left rotation transformation for the third module. The function corresponding to the extra output bit is $x_2 \quad x_4 \quad x_6 \quad ... \quad x_n$ where x_i's are the input bits.

For example, consider a network N implementing any arbitrary logic function with $n = 4$ inputs, x_1, x_2, x_3 and x_4. We add another output function $g = x_2 \quad x_4$ to N. Consider a TMR system where the input registers of the first and the second modules contain the true and complemented values of the variables, respectively, while the input register of the third module contains variable values under a left rotation transformation. Let us consider an input-register CMF such that the first bits of the input registers of module 1 and module 3 are stuck at 1. The first bit position of the input register of module 1 contains the x_1 value, while that of the third module contains x_2 value. Consider any input combination with $x_1 = 0$ and $x_2 = 0$. The *g* output of the first module is x_4. However, the *g* output of the third module is x_4'. Hence, the output *vectors* from the first and the third modules can never match and the two modules affected by the CMF can never produce the same output vector. Similar case happens for other input-register CMFs. Thus, either the TMR produces correct outputs or generates the *ERROR* signal when an incorrect output vector is produced.

Theorem 3: A function with an odd number (*n*) of input bits can be made fault-secure with respect to the common-mode faults, by adding a maximum of two extra output bits, if we apply the left rotation transformation for the third module. The function corresponding to the extra output bit can be $x_1 \quad x_3 \quad x_5 \quad ... \quad x_n$. The second output function can be equal to x_1 or x_n.

7. REDUNDANCY AND EXTENDED INPUT-REGISTER-CMF MODEL

In this section, we extend the input-register-CMF model of Sec. 3. For simplicity and space constraints, we explain our technique to design duplex systems that are fault-secure against the extended input-register-CMFs. However, the whole discussion can be extended for TMR systems.

Under the extended fault model, we relax the constraint that only the corresponding bit-positions of the input registers can be stuck. For example, under the extended IR-CMF model, the second bit position of the input register of the first module and the fifth bit position of the input register of the second module can be stuck at the same time. The rationale behind this is that, the model of Sec. 3 mainly targets systems where the registers are organized in register files according to the RAM model. However, this may not be true for registers in many ASICs that are synthesized, placed and routed using conventional CAD tools. In this case, a radiation source can cause multiple upsets in different bit positions of multiple registers depending on the physical proximities of the storage elements of the different registers. This information is not usually available during the design phase. Also, it may be impractical to generate layouts on the basis of vulnerability to radiation upsets. This is because, area and performance of the final chip is strongly dependent on its layout. Moreover, even for register-file-based designs, the actual adjacencies of the different bit-positions of the registers in the layout-level may be different from the logical view of the register-file (Fig. 3.1). Hence, during the design phase we must guarantee that the system is fault-secure against these extended input-register-CMFs.

Table 7.1. An example logic function

a b c	f_1 f_2
0 0 0	0 0
0 0 1	0 0
0 1 0	0 1
0 1 1	1 0
1 0 0	1 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 0

Consider the example logic function of Table 7.1. Suppose we want to design a duplex system for this logic function. Let us call the two modules of the duplex system N_1 and N_2. The first module (N_1) of the duplex system is obtained by normal synthesis of the function — its input register contains the values of *a*, *b* and *c* in the three flip-flops. For the second module, we want to find a *good* transformation T of the inputs of the logic function. The outputs of the network implementing the transformation T will be connected to the input register of the second module. Finally, the second module will

take inputs from its input register and produce outputs. The scheme is depicted in Fig. 7.1. The transform T can be specified in the following way. For each combination *i* of *a*, *b* and *c*, transformation T produces a particular *symbolic output* X_i. The actual value of X_i, in terms of 1s and 0s, depends on the logic function implemented by T. Table 7.2(a) shows the specification of T.

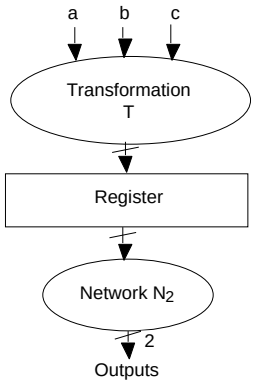

Figure 7.1. The basic scheme for the third module

If we examine Table 7.1, we find that input combinations 000 and 001 produce the same output vector (00). Hence, the transformation T can produce the same output (and hence, the same output symbol) in response to input combinations 000 and 001. Two output symbols X_i and X_j in the specification of T (Table 7.2(a) here) are *equivalent* if and only if input combinations *i* and *j* produce the same output vector in the truth table of the logic function to be implemented (Table 7.1 here).

Table 7.2. Transformation T (a) No equivalence (b) Equivalence (c) Specification of N_2

(a)

Inputs a b c	Output Symbols
0 0 0	X_0
0 0 1	X_1
0 1 0	X_2
0 1 1	X_3
1 0 0	X_4
1 0 1	X_5
1 1 0	X_6
1 1 1	X_7

(b)

Inputs a b c	Output Symbols
0 0 0	Z_1
0 0 1	Z_1
0 1 0	Z_2
0 1 1	Z_3
1 0 0	Z_4
1 0 1	Z_4
1 1 0	Z_2
1 1 1	Z_3

(c)

Input	Output
Z_1	00
Z_2	01
Z_3	10
Z_4	11

For Table 7.2(a), we find that X_0 and X_1 are equivalent. Hence, we replace them by the symbol Z_1 in Table 7.2(b). Similarly, X_2 and X_6 are equivalent and are replaced by the symbol Z_2. Equivalent symbols X_3 and X_7 are replaced by the symbol Z_3; and, X_4 and X_5 by Z_4 in Table 7.2(b). Note that, minimizing the number of output symbols through equivalence does not always generate the minimum logic, as has been observed in [Hartmanis 62]. The problem is analogous to the state minimization problem in logic synthesis.

Finding the transformation T is equivalent to encoding the Z_i's of Table 7.2(b) using 1s and 0s in the presence of some constraints that will make the duplex system fault-secure or self-testing with respect to the extended input-register-CMFs. The problem can be modeled as a constrained output encoding problem for the specification in Table 7.2(b). In the next paragraph, we discuss about the constraints to be satisfied.

Module 1

	Input Combination	Output of First Module
Fault-free	000	$00 = N_2(Z_1)$
Faulty: Bit 1 stuck-at 1	100	$11 = N_2(Z_4)$

Module 2

	Input combination	Output of third module
Fault-free	$T(000) = Z_1$	$N_2(Z_1)$
Faulty: (Any single bit stuck)	Anything except Z_4	Anything except $N_2(Z_4)$

Figure 7.2. Illustration of the generation of distance constraints

Let us consider an extended IR-CMF, due to which, the first bit of the input register of the first module is stuck-at 1 and any arbitrary bit of the input register of the second module is stuck at a particular value. When input combination 000 is applied, the first module has 100 at its input and produces 11 (Table 7.1). For a fault-secure system, we want that the second module must not produce 11. Thus, for the second module (N_2), the code corresponding to Z_1 must not get changed to the code of Z_4 due to the presence of a single stuck-at fault at any bit position of the input register of the third module. This can be written as a *distance* constraint: $d(Z_1, Z_4) > 1$. Here, $d(Z_1, Z_4)$ is the notation for the distance between Z_1 and Z_4. This is illustrated in Figure 7.2. The algorithm for generating the distance constraints is shown below.

Algorithm: Generate Distance Constraints
For each CMF
    For each input combination *i*
    Suppose *i* changes to input combination *j* in the presence of the CMF
    If $Z_i = Z_j$ no distance constraint generated
    else Distance constraint $d(Z_i, Z_j) > 1$ is generated

For our current example, the distance constraints generated are: $d(Z_1, Z_2) > 1$, $d(Z_1, Z_3) > 1$, $d(Z_1, Z_4) > 1$, $d(Z_2, Z_3) > 1$ and $d(Z_2, Z_4) > 1$. The existence of an encoding satisfying all the constraints is always guaranteed by using an extra parity bit. The constrained output encoding can be accomplished by modifying some exact [Devadas 91] or heuristic encoding techniques [Lin 90]. Table 7.3 shows an encoding for the symbols in Table 7.2(b) that satisfies all these distance constraints.

Table 7.3. Codes for fault-secure duplex system

Symbol	$c_1\ c_2\ c_3$
Z_1	0 0 1
Z_2	0 1 0
Z_3	1 1 1
Z_4	1 0 0

Table 7.4. The final truth table for N_2

$c_1\ c_2\ c_3$	$o_1\ o_2\ o_3$
0 0 0	- - 1
0 0 1	0 0 0
0 1 0	0 1 0
0 1 1	- - 1
1 0 0	1 1 0
1 0 1	- - 1
1 1 0	- - 1
1 1 1	1 0 0

Table 7.5. Encoding for self-testing duplex system

Symbol	$c_1\ c_2$
Z_1	0 0
Z_2	0 1
Z_3	1 0
Z_4	1 1

In Table 7.3, we have used 3 bits to encode four symbols. However, there are 8 possible binary combinations with the 3 bits — there are 4 *don't care* combinations. We must ensure that the fault-secure property does not get violated because of these don't cares. For example, suppose we apply input combination 000 to the system. The register corresponding to the second module should contain 001. Consider a CMF, as a result of which the first bits of the input registers of the two modules are stuck-at 1. In response to input combination 000, the first module sees 100 at its inputs and produces output 11. Network N_2 sees 101 at its input. However, 101 is a *don't care* combination. We must ensure that N_2 does not produce 11 in response to 101.

In general, since $d(Z_1, Z_2) > 1$, $d(Z_1, Z_3) > 1$, $d(Z_1, Z_4) > 1$ had to be satisfied while performing output encoding, network N_2 must not produce 11, 01 or 10 in response to any combination of c_1, c_2 and c_3 at a distance 1 from the code of Z_1. 101 is such an input combination which is at unit distance from 001, the code of Z_1. However, 101 is at unit distance from the code of Z_4 (100) and since $d(Z_4, Z_1) > 1$, network N_2 must not produce 00 in response to 101. Hence, we need an extra output combination. Since there are two output bits, there can be a maximum of 4 output vectors. So we need an extra output bit. We can perform similar computations for the remaining don't care combinations. The truth table of N_2 is shown in Table 7.4.

Note that, for this particular example, we did not have enough choices for the output vectors that the don't care combinations on the input variables should produce. *It is not necessarily true that we always have to add an extra output bit*. In that case, we

can use a Boolean relation technique [De Micheli 94] to solve the problem of assigning output vectors to the don't care combinations on input variables.

However, if we want to make the above system self-testing, instead of fault-secure, with respect to the extended input-register CMFs, we are less constrained so far as the distance constraints are concerned. In that case, can perform the encoding as shown in Table 7.5. Note that, for the self-testing property, we can accomplish the encoding using only two bits (instead of three bits required to achieve the fault-secure property). Since we have four symbols to be encoded and we use only two bits, we do not have to worry about the don't care combinations. The logic complexity is lower for achieving the self-testing property compared to the fault-secure property. The flow chart for the encoding technique to generate a fault-secure or self-testing duplex system is shown in Fig. 7.2.

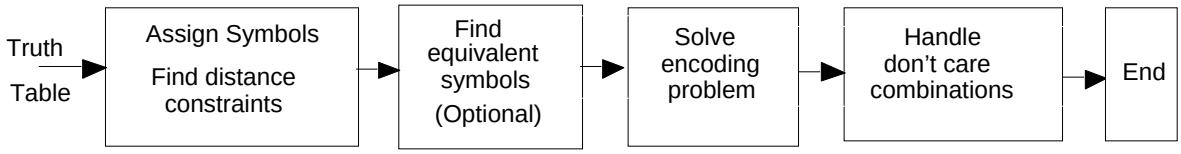

Figure 7.3. The flow chart for the technique

As shown in Fig. 7.3, given the truth table, we assign symbols corresponding to each input combination and find the distance constraints to make the final duplex system fault-secure or self-testing with respect to the CMFs. Next, we can find the equivalent symbols and reduce the number of symbols to be encoded. This step is optional since it may not be true that minimizing the number of symbols always minimizes the complexity of the resulting logic. Next we solve the encoding problem. If there are some don't care combinations that arise due to the number of encoding bits, we need to assign output vectors to these don't care input combinations, as discussed earlier in this section, so that the distance constraints are not violated. Our discussions in this section can be easily extended for TMR systems protected against extended input-register CMFs.

8. SIMULATION RESULTS

In this section, we present simulation results. We applied the technique described in Sec. 7 to some duplex systems for MCNC benchmark functions. For solving the encoding problem, shown in the flow chart of Fig. 7.2, we modified JEDI [Lin 90], an existing FSM state encoding tool, to incorporate the distance constraints that are generated by our technique. For simplicity, we allowed equivalent input combinations (i.e., input combinations which produce the same output vector) to have the same code. We performed encoding with two different objectives:

1. Perform encoding in order to make the duplex system *fault-secure* with respect to the extended input-register-CMFs.
2. Perform encoding in order to make the duplex system *self-testing* with respect to the extended input-register-CMFs.

For synthesizing the logic circuits, we used the *Sis* tool [Sentovich 92]. For technology mapping purposes, we used the LSI Logic G10p library [LSI 96]. In Table 8.1, for each benchmark, we present a comparison of the areas of conventional duplex systems obtained through direct replication and the systems synthesized using our technique presented in Sec. 7.

Table 8.1. Area Comparison

Circuit Name	# Inputs	# Outputs	Normal Duplex Area	Fault-Secure Duplex		Self-testing Duplex	
				Area	% Overhead	Area	% Overhead
example1	15	9	500	480	-4	428	-14
c17	5	2	80	107	33	92	15
clip	9	5	846	1002	18	752	-11
misex1	8	3	340	307	-10	284	-16

It may be noted, that for some of the cases (with negative overheads), the duplex systems which provide protection against the extended input-register CMFs consume *lesser* area than a conventional duplex systems with two replicated modules. In our experiments, we generated the transformation T first. Then, on the basis of the encoding, we generated the logic network N_2. However, an interesting problem will be to find *good* encodings that simultaneously optimize the encoding logic and the logic producing the final outputs and satisfy the given constraints for building fault-secure or self-testing duplex systems. This is a *new* encoding problem in logic synthesis.

9. SUMMARY AND CONCLUSIONS

It is a well-known fact that redundant systems are prone to common-mode failures. This paper shows, for the first time, that it is possible to develop fault-models for common-mode failures (CMFs) in redundant systems. The CMF models provide new opportunities to design redundant systems so that data-integrity is maintained during system operation in the presence of the modeled CMFs. This paper also presents one such design technique. Our simulation results demonstrate that redundant systems designed using our technique require comparable or less area than conventional redundant systems that are not protected against CMFs. Moreover, our design technique gives rise to new and very interesting problems that may prove to be extremely useful for logic synthesis.

10. ACKNOWLEDGMENTS

This work was supported by Defense Advanced Research Projects Ajency (DARPA) under Contract No. DABT63-97-C-0024. The authors would like to thank Dr. Nirmal Saxena and Philip Shirvani of Stanford CRC for their comments.

11. REFERENCES

[Ashar 91] Ashar, P., A. Ghosh and S. Devadas, "Boolean Satisfiability and Equivalence Checking using general Binary Decision Diagrams," *Proc. ICCD*, pp. 259-264, 1991.

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

[Choi, 93] Choi, G. S., R. Iyer and D. Saab "Fault behavior dictionary for simulation of device-level transients," *Proc. ICCAD*, pp. 6-9, 1993 pp. 6-9.

[De Micheli 94] De Micheli, G, *Synthesis and Optimization of Digital Circuits*,McGraw-Hill, Inc., 1994.

[Devadas 91] Devadas, S. and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four Level boolean Minimization," *IEEE Trans. on CAD*, 10(1), pp. 13-27, Jan. 1991.

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

[Hartmanis 62] Hartmanis, J., et. al., "Some dangers in state reduction of sequential machines," *Information and Control*, pp. 252-260, September, 1962.

[Lala 93] Lala, J. H. and R. E. Harper, "Reducing the probability of common-mode failure in the fault-tolerant parallel processor," *Proc. IEEE/AIAA Digital Avionics Systems Conference*, pp. 221-230, 1993.

[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.

[Liden 94] Liden, P., et. al., "On Latching Probability of Particle-Induced Transients in Combinational Networks," *Proc. FTCS*, pp. 340-349, 1994.

[Lin 90] Lin, B. and A. R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," *VLSI 89*, pp. 187-196, Elsevier Science Publishers, 1990.

[McCluskey 86] McCluskey, E. J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, Eaglewood Cliffs, NJ, USA, 1986.

[Mitra 00] Mitra, S. and E. J. McCluskey, "Word-Voter: A New Voter Design for Triple Modular Redundant Systems," *Proc. VLSI Test symposium*, 2000, To appear.

[Reed 97] Reed, R., et. al., "Heavy ion and proton-induced single event multiple upset," *IEEE Trans. on Nuclear Sience*, Vol. 44, No. 6, pp. 2224-2229, July 1997.

[Rimen 94] Rimen, M., J. Ohlsson and J. Torin, "On microprocessor error behavior modeling," *Proc. FTCS*, pp. 76-85, 1994.

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

[Stroud 94] Stroud, C. E., "Reliability of Majority Voting Based VLSI Fault-Tolerant Circuits," *IEEE Trans. on VLSI*, vol. 2, no. 4, pp. 516-521, December, 1984.

[Trivedi 82] Trivedi, K.S., *Probability and statistics with reliability, queuing, and computer science applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[Von Neumann 56] Von Neumann, J., "Probabilistic Logics and the synthesis of reliable organisms from unreliable components," *Automata Studies, Ann. of Math. Studies*, no. 34, C. E. Shannon and J. McCarthy, Eds., Princeton University Press, pp. 43-98, 1956.

[Zhang 93] Zhang, H, "Sato: A Decision Procedure for Propositional Logic," *Automated Reasoning Newsletters*, 22, 1993. Package available from the Department of Computer Science, University of Iowa (ftp: herky.cs.uiowa.edu).

APPENDIX

A.1. FINDING TRANSFORMATIONS USING BOOLEAN SATISFIABILITY

In this section, we explain the technique of finding suitable transformations for self-testability of TMR systems against input-register-CMFs (Sec. 5) in details. By module N, we will refer to the first module of the logic circuit — true values of the input variables (X, Y, Z, etc.) are applied to the inputs of module N. By module N_1, we mean the second module to whose inputs complemented values (X′, Y′, Z′, etc.) are applied. For the third module (N_2), we apply a transformed version of the input to module N, the transformation function being T. We assume that T is a bijection. Here we will describe a technique to find T using Boolean Satisfiability.

Since the module N has n inputs, there are 2^n possible input combinations for N. Consider an input combination m. Let m_i denote the i^{th} bit of m. Let us suppose that the first bit of m is 0; i.e., $m_1 = 0$. If we assume a common-mode fault f which results in a stuck-at 1 fault in the first-bit of all the three input registers, then due the presence of f, network N will see an input combination q which has a 1 in the first bit and the remaining bits of q are the same as the corresponding bits of m. We can write $q = m + 100...0$, where '+' means a bit-wise OR operation. Since N_1 sees complemented input values and the first bit in the *complement* of m is 1, N_1 will produce the correct output vector v_1.

Now, if N produces the correct output vector v_1 in response to q, then we will get a correct vector at the system output. But if N produces an incorrect output vector v_2 in response to q, then we want that N_2 should either produce the correct output vector or it should produce an *incorrect* output vector *different* from the one produced by N. Note that, N_2 sees the combination T(m) + 100...0 at its input, where T is the transformation function that we want to determine. Let r is an input combination, in response to which N produces an output vector v_3 , $v_3 \neq v_1$, $v_3 \neq v_2$. If the following relation holds, then we will be able to detect the CMF under consideration.

$$T(m) + 100...0 = T(r).$$

This means that T(r) and T(m) should differ only in the first bit. The above relationship can be looked upon as a constraint that the transformation T should satisfy. We can call this constraint $C_{1,\,1,\,3}^{1,\,m,\,r}$. The first 1 in the subscript means that we are considering a common-mode fault of type stuck-at 1. The remaining two symbols in the subscript means that modules 1 and 3 produce wrong output vectors and module 2 produces the correct output vector. The 1 in the superscript indicates the bit position and

m and r have been defined before. To write the constraint, we introduce $n2^n$ boolean variables $w_{i,j}$ ($1 \le i \le 2^n$, $1 \le j \le n$). Here $w_{i,j}$ is a boolean variable representing the j^{th} bit of T(i). The constraint is:

$$(\sim w_{m,1} \quad w_{r,1} \quad (w_{m,2} \equiv w_{r,2}) \quad \ldots \quad (w_{m,n} \equiv w_{r,n}))$$

In the above formula, ≡ stands for equality which can be expresed as XNOR operation. The above formula says that the first bit of T(m) and T(r) should be 0 and 1, respectively. Also, T(m) and T(r) must differ *only* in the first bit position. Note that, if $m_1 = 1$, then also we can write a similar constraint because, in that case module N_1 will be affected.

Now we consider the constraints generated by a common-mode fault of type stuck-at 0. Consider a common-mode fault f which results in a stuck-at 0 fault in the first-bit of all the three input registers. Let us suppose that we apply an input combination m, where $m_1 = 1$. Due the presence of f, network N will see an input combination q which has a 0 in the first bit and the remaining bits of q are the same as the corresponding bits of m. Hence, $q = m$ & 011...1, where '&' means a bitwise AND operation. The input of N_1 is not going to be changed due to the presence of this common-mode fault — N_1 will produce the correct output vector v_1. Now, if N produces the correct output vector v_1 in response to q, then we will get a correct vector at the system output. But if N produces an incorrect output vector v_2 in response to q, then we want that N_2 should either produce the correct output vector or it should produce an *incorrect* output vector different from the one produced by N. Note that, N_2 sees the combination T(m) & 011...1 at its input, where T is the transformation function that we want to determine. If r is an input combination, in response to which N produces an output vector v_3 , $v_3 \neq v_1$, $v_3 \neq v_2$, then we may want the following relation to hold.

$$T(m) \,\&\, 011\ldots1 = T(r).$$

We call this constraint $C_{0,1,3}^{1,m,r}$. The subscript 0 means that we are considering a common-mode fault of type stuck-at 0. The constraint $C_{0,1,3}^{1,m,r}$ is:

$$(w_{m,1} \quad \sim w_{r,1} \quad (w_{m,2} \equiv w_{r,2}) \quad \ldots \quad (w_{m,n} \equiv w_{r,n}))$$

We present Algorithm 1, which generates the set of constraints which the transform T must satisfy in order to detect a given CMF.

Algorithm 1
Input: A CMF involving bit *i*, where bit *i* 's of any pair of input registers are stuck-at *j*
Truth table of combinational network N
Output: A constraint, which when satisfied, detects the fault
Steps:
1. For each input combination (*m*)
2. Let N produce output vector v_1 in response to *m*
3. Find *q* which differs from *m* only in bit position *i*
3. If N produces output vector v_1 in response to *q* then continue
4. Let N produce output vector v_2 in response to *m*
5. For each input combination (*r*)
6. Let N produce an output vector v_3 in response to *r*
7. If ($v_1 = v_3$) or ($v_2 = v_3$) continue
8. If $m_i = j$, generate constraint $C_{j,\,2,\,3}^{i,\,m,\,r}$ else generate $C_{j,\,1,\,3}^{i,\,m,\,r}$
9. Constraint $C_{j,\,1,\,3}^{i}$ is the OR of all the $C_{j,\,1,\,3}^{i,\,m,\,r}$'s generated
10. Constraint $C_{j,\,2,\,3}^{i}$ is the OR of all the $C_{j,\,2,\,3}^{i,\,m,\,r}$'s generated

Once we generate the constraints according to Algorithm 1, we perform AND-ing of all the constraints $C_{j,\,1,\,3}^{i}$ and $C_{j,\,2,\,3}^{i}$ where $j = 0, 1$ and $i = 1, ..., n$, where n is the number of input bits. The resulting constraint is called *detectability constraint*. However, we need another set of constraints which ensure that the transformation T is a bijection. These constraints are called $D_{i,\,j}$, $i \neq j$, which stands for the *distinctness* constraint between input combinations T(*i*) and T(*j*). $D_{i,\,j}$ can be written as follows:

$$(\sim(w_{i,\,1} \equiv w_{j,\,1}) \quad \sim(w_{i,\,2} \equiv w_{j,\,2}) \quad ... \quad \sim(w_{i,\,n} \equiv w_{j,\,n}))$$

The constraint $D_{i,\,j}$ tells us that T(*i*) and T(*j*) must differ in at least one bit. This is needed because we assumed that T is one-to-one. The conjunction of all these distinctness constraints and the detectability constraint gives the final formula that we have to satisfy. Note that, the way the above constraints are generated, any CMF affecting input registers of module 1 and module 3 or input registers of module 2 and module 3 will be detected. Also note that a single-bit CMF affecting the input registers of module 1 and module 2 will be tolerated.

Table A.1.1. An example logic function

x y z	f_1 f_2
0 0 0	0 0
0 0 1	0 0
0 1 0	0 1
0 1 1	1 0
1 0 0	1 1
1 0 1	1 1
1 1 0	0 1
1 1 1	1 0

Now, we explain the constraint generation algorithm with the help an example. Consider the logic function shown in Table A.1.1.

Let us name the eight input combinations in the following way: a = 000, b = 001, c = 010, d = 011, e = 100, f = 101, g = 110 and h = 111. Consider the common-mode fault which results in a stuck-at 1 fault in the first bit (bit 1) position of the input registers. Now, we apply Algorithm 1 in order to obtain the constraints. Consider the input combination a (000). Since a_1 = 0, q = 100. Network N (the first module) produces output vector v_1 = 00 in response to a (000) and vector v_2 = 11 in response to q (100). Now, c = 010 is an input combination that produces an output vector v_3 = 01 that is different from 00 or 11. Hence, we generate the constraint $C_{1,1,3}^{1,a,c} = (\sim w_{a,1} \quad w_{c,1} \quad (w_{a,2} \equiv w_{c,2}) \quad (w_{a,3} \equiv w_{c,3}))$. Input combination g = 110 produces output vector 01 that is different from 00 and 11. So we also generate the constraint $C_{1,1,3}^{1,a,g} = (\sim w_{a,1} \quad w_{g,1} \quad (w_{a,2} \equiv w_{g,2}) \quad (w_{a,3} \equiv w_{g,3}))$. In a similar way, we generate constraints $C_{1,1,3}^{1,a,d}$ and $C_{1,1,3}^{1,a,h}$. Now, if we consider the input combination 001 (b), we find that N produces output vector v_1 = 00 in response to b. Now, q = 101. In response to q (101), the network N produces output vector v_2 = 11. Hence, we generate constraints $C_{1,1,3}^{1,b,c}$, $C_{1,1,3}^{1,b,g}$, $C_{1,1,3}^{1,b,d}$ and $C_{1,1,3}^{1,b,h}$. All these constraints correspond to the detection of a CMF of type stuck-at 1, affecting the first bit position of the input registers of module 1 and 3. No constraint is generated for input combination c (010), because network N produces the same output vector for input combinations 010 and 110. Similarly, no constraint will be produced for input combination d (011). Next consider the input combination e = 100. In this case, q is 000 since it differs from 100 only in the first bit. For input combinations 100 and 000, network N produces output vectors 11 (v_1) and 00 (v_2), respectively. Now, we generate the following constraints: $C_{1,2,3}^{1,e,c}$, $C_{1,2,3}^{1,e,g}$, $C_{1,2,3}^{1,e,d}$ and $C_{1,2,3}^{1,e,h}$. These constraints correspond to the detection of a CMF of type stuck-at 1, affecting the first bit position of the input registers of module 2 and 3. Note that, the CMF of type stuck-at 1, affecting the first bit position of the input registers of all the three modules is automatically covered. Similar constraints are generated for the remaining input combinations.