Hierarchical Fault Collapsing for Logic Circuits

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Raja Kiran Kumar Reddy Sandireddy

Certificate of Approval:

Charles E. Stroud
Professor
Electrical and Computer Engineering

Vishwani D. Agrawal, Chair
Professor
Electrical and Computer Engineering

Victor P. Nelson
Professor
Electrical and Computer Engineering

Stephen L. McFarland
Dean
Graduate School

Hierarchical Fault Collapsing for Logic Circuits

Raja Kiran Kumar Reddy Sandireddy

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama

13 May 2005

Hierarchical Fault Collapsing for Logic Circuits

Raja Kiran Kumar Reddy Sandireddy

Signature of Author

Date

Copy sent to:

Name Date

Vita

Raja K. K. R. Sandireddy, son of S. Rami Reddy and B. Suseelamma, was born in Kurnool, Andhra Pradesh, India. He graduated from St. Joseph Institutions, Kurnool in 1998. He earned the degree Bachelor of Technology in Electronics and Communication Engineering from Regional Engineering College (now National Institute of Technology), Warangal, India in 2002.

Thesis Abstract

Hierarchical Fault Collapsing for Logic Circuits

Raja Kiran Kumar Reddy Sandireddy

Master of Science, 13 May 2005
(B.Tech., Regional Engineering College, Warangal, India, 2002)

93 Typed Pages

Directed by Vishwani D. Agrawal

The process of grouping stuck-at faults into equivalence or dominance classes and then selecting one representative for each class is known as fault collapsing. There have been several techniques to collapse the faults structurally and functionally. We present a new method that determines all functional dominances between the faults of a circuit using redundancy identification and then collapses faults. This procedure produces the smallest possible equivalence and dominance collapsed sets. Our method is more efficient than other functional collapsing algorithms, but is still expensive in CPU time. Therefore, we recommend its application only to smaller sub-circuits that may be embedded in larger circuits.

Another contribution of this research is a thorough examination of the fault equivalence and dominance relations for multiple output circuits. The conventional definition for equivalence says that "two faults are equivalent if and only if the corresponding faulty circuits have identical output functions." This definition is extended for multiple output circuits as "two faults of a Boolean circuit are called diagnostically equivalent if and only if the functions of the two faulty circuits are identical at each

output." Alternatively, "if all tests that detect a fault also detect another fault, not necessarily at the same output, then the two faults are called detection equivalent." The definitions for fault dominance follow on similar lines.

A program implements our novel algorithm based on redundancy identification to find the complete equivalence and dominance collapsed sets using diagnostic and detection collapsing definitions. When applied to a 4-bit ALU it collapses the total fault set of 502 faults to 253 and 155, respectively, according to diagnostic equivalence and dominance. The collapsed sets have 234 and 92 faults, respectively, for detection equivalence and dominance. In comparison, the conventional structural equivalence and dominance collapsing results in 301 and 248 faults, respectively.

Functional fault collapsing, using the proposed algorithm, is done for standard cells and other logic blocks and the collapse data is saved as library information. Using the redundancy-based fault collapsing for a full adder library cell, we hierarchically collapse faults in a 64-bit adder to sets of 1538 equivalence and 768 dominance collapsed faults. In comparison, a flattened 64-bit adder circuit collapses into 2306 and 1794 equivalence and dominance collapsed sets, respectively. It is also shown that hierarchical collapsing results in an order of magnitude reduction in the CPU time for very large circuits. A flattened 1024-bit adder requires 39.9 seconds for structural collapsing while the same circuit, if described hierarchically as an interconnect of 1024 full adders, takes only 3.60 seconds. The 1024-bit adder can also be hierarchically built using four 256-bit adders, which are built using four 64-bit adder, and so on and this circuit takes only 2.31 seconds to collapse faults. In conclusion, hierarchical collapsing saves CPU time and potentially provides more collapsing.

Acknowledgments

I would like to acknowledge the support and guidance from Dr. Vishwani D. Agrawal, whose suggestions and directions have a major influence on all aspects of this thesis. It was a great pleasure for me to undergo this learning experience. I thank Dr. Charles E. Stroud and Dr. Victor P. Nelson for being on my committee and providing me valuable inputs. I thank Dr. Fa Foster Dai and Dr. Richard C. Jaeger who supported me during the first year of my study at Auburn. I am thankful to all my friends at Auburn for being the surrogate family during my Masters. I cannot end without thanking my parents and sister, on whose constant encouragement and love I have relied throughout my life.

Style manual or journal used LaTeX: A Document Preparation System by Leslie Lamport (together with the style known as "aums").

Computer software used The document preparation package TeX (specifically LaTeX) together with the departmental style-file `aums.sty`. The plots and images were generated using MATLAB®and XFig.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The number of faults to be tested in a circuit has a strong influence on the costs incurred in the testing process. So, faults are collapsed based on equivalence and dominance relations. The time required for fault simulation is directly proportional to the number of faults. Also, with a smaller collapsed set, we can come up with fewer test vectors and in case of diagnosis, better resolution may be possible. The time taken for collapsing should not offset the savings achieved during test generation and fault simulation.

The classical definitions of equivalence and dominance have been analyzed and redefined as diagnostic equivalence and diagnostic dominance. The classical definitions of equivalence and dominance, when interpreted in a different way, especially for multiple output circuits, lead to the terms detection equivalence and detection dominance in this thesis. It is observed that fault simulation inherently uses the "detection" definitions, while engineers assume the "diagnostic" definitions. This observation motivated the work presented in this thesis.

1.1 Problem Statement

The problem solved in this thesis is: *Find an optimal collapsed fault set for a large circuit using an acceptable amount of computer time.*

1.2 Contributions of Thesis

We have analyzed the definitions of equivalence and dominance for circuits with more than one output. Using these definitions, a novel redundancy identification based algorithm is presented that finds all functional fault dominance relations in a given circuit. Since redundancy identification takes time exponential in the circuit size, the procedure is recommended only for small circuits.

In order to take advantage of the better collapse ratios obtained with functional collapsing, while keeping the collapse time in check, a hierarchical fault collapsing technique is used. This technique, which was proposed earlier [46, 75], is explained in great detail in Chapter 5. The hierarchical fault collapsing technique, when used for structural collapsing, takes less time than the flat level collapsing. In addition, when functional collapsing is applied to small sub-circuits, the hierarchical procedure results in smaller collapsed sets while retaining the CPU time advantage.

This work has been accepted at Design, Automation and Test in Europe (DATE) 2005 Conference [78].

1.3 Organization of Thesis

The outline of this thesis is as follows. In Chapter 2, we discuss the general concepts of testing and background of faults and fault collapsing. In Chapter 3, the previous work on functional collapsing is discussed. Also, hierarchical fault collapsing is explained and the work of two previous papers on hierarchical fault collapsing is detailed in Chapter 3. The original contributions of this work start from Chapter 4. The conventional definitions of equivalence and dominance are analyzed and extended

to multiple output circuits in Chapter 4. A novel algorithm has been proposed to find dominance and equivalence collapsed fault sets, based on the definitions for multiple output circuits. The hierarchical fault collapsing technique is implemented in Chapter 5 and results for sample circuits are presented. Finally, the conclusions and future work are discussed in Chapter 6.

Chapter 2

Background

In engineering, models bridge the gap between the physical reality and mathematical abstraction [24]. Models allow the development and application of analytical tools and are therefore essential in design. Our focus in this work is on the models used in the testing of digital circuits. Testing is the process of verifying that the manufactured circuit has the intended structure. Testing has become a dominating cost in the design process, amounting to 30% or more of the total cost [24]. The most important models in testing are those of faults. Physical faults in an electronic system can be modeled as either logic faults or delay faults depending on whether the fault affects the logic function or the operating speed of the system. This thesis concentrates on the logical fault models.

There are advantages of modeling logical faults [1, 24]. First, the problem of fault analysis can be treated at the functional level. Its complexity is greatly reduced because many physical faults can be represented by the same logical fault. Second, few other fault models are technology-independent. The test and diagnosis methods for such models remain valid with the changes in technology. Third, tests derived for logical faults may be effective for detecting many physical faults whose effects on the circuit behavior are not completely understood or are too complex to be analyzed [50].

2.1 Stuck-at Faults

Logical faults are modeled as stuck-at faults. A stuck-at fault forces a fixed (0 or 1) value to a signal line in the circuit, where a signal line can be an input or an output of a logic gate or flip-flop [24]. The stuck-at fault can be $stuck-at-1$ (s-a-1 or sa1) or $stuck-at-0$ (s-a-0 or sa0). This model is simple and technology-independent. In general, a circuit can have many stuck-at faults. Since a line can be s-a-0, s-a-1, or fault-free, a circuit with n signal lines can have $3^n - 1$ possible stuck line combinations [24]. All combinations except one, having all lines in fault-free states, are counted as faults. So, even for circuits of moderate size, i.e., for a moderate value of n, an enormously large number of multiple stuck-at faults have to be considered. Therefore, it is a common practice to consider single stuck-at fault model, which is the most widely used logical fault model [1, 24]. A circuit with n lines can have at most $2n$ single stuck-at faults [24]. The number of faults that are considered for testing is reduced by fault collapsing as explained in the next section.

It is believed that Eldred's 1959 paper [35] laid the foundation for the stuck-at fault model though that paper did not make any mention of the stuck-at fault. The term "stuck-at fault" appeared in the paper by Galey, Norby, and Roth in 1961 [38].

A typical test generation process targeting a given fault set for a combinational circuit is explained here. A random pattern generator is used to generate random input patterns. The circuit after introducing a fault is simulated with those random input patterns. This is done for all the faults in consideration and it is called fault simulation. The faults that are found to be detectable are dropped and this procedure is called fault dropping. These steps are repeated until a preset percentage, typically

$60 - 80\%$, of faults is detected. This phase of the test generation process is called random pattern generation [5]. After the random phase, from the remaining fault set, a target fault is selected and a test that detects this fault is found using an algorithmic test pattern generator. That test is then simulated to find the other faults it detects, which are then removed from the remaining fault set. This procedure is continued until all faults are targeted or detected. This second phase is called deterministic pattern generation. The faults left undetected are either redundant faults for which no test exists, or aborted faults that could not be detected due to CPU time limit. A circuit is called redundant if and only if it contains some stuck-at fault that is redundant [49].

For a given fault set, the problem of generating a minimum sized test set is considered NP-hard [57]. But, the same problem reduces to polynomial complexity in the size of the circuit for certain classes of combinational logic circuits [17, 25, 36, 51, 71]. As the complexity of the circuit under test increases, the number of faults increases and thereby the effort to generate a test set increases at a greater rate [41]. One way of reducing the time for test generation and fault simulation is fault collapsing.

2.2 Fault Collapsing

Fault collapsing can be classified into two types - equivalence collapsing and dominance collapsing. Two faults are called equivalent if and only if the corresponding faulty circuits have identical output functions [24]. Two faults are said to be distinguishable if the faulty circuits resulting from the two faults have different responses

for at least one set of input conditions [79]. Equivalent faults are indistinguishable because they cannot be distinguished at the primary outputs by any input vector. The set of all faults in a circuit can be partitioned into equivalence sets, such that all faults in a set are equivalent to each other. Equivalence collapsing is the process of fault partitioning and selecting one fault from each equivalence set. The fault set thus obtained is called equivalence collapsed set. For an n-input OR gate, the total number of stuck-at faults is $2n + 2$. All single s-a-1 faults on the inputs and output of the OR gate are equivalent. Thus, using equivalence collapsing, the number of stuck-at faults is reduced to $n + 2$.

Another form of collapsing that can further reduce the fault set size is dominance fault collapsing. If all tests of fault f_1 detect another fault f_2, then f_2 is said to dominate f_1. The two faults are also called "conditionally" equivalent with respect to the test set of f_1 [24]. In dominance collapsing, all dominating faults in an equivalence collapsed set are left out retaining their respective dominated faults. For the OR gate, the output s-a-0 fault dominates a single s-a-0 fault on any input and hence the output s-a-0 is left out of the dominance collapsed set. Thus, dominance fault collapsing reduces the number of faults for an n-input OR gate to $n + 1$. Dominance collapsing always results in a smaller set than the equivalence collapsed set.

Dominance is a more basic property than equivalence. When two faults dominate each other, then they are equivalent. So, if we know all the dominance relations, then we can find the equivalence relations too. In the equivalence collapsed set, when a fault is not detected, the status of the entire set of faults that is equivalent to it is

known. Such is not the case in the dominance collapsed set. Still there are advantages of using the latter for ATPG (Automatic Test Pattern Generation).

Fault collapsing can also be classified as structural collapsing and functional collapsing. Structural collapsing depends on the effect of a fault on the circuit graph or structure, while functional collapsing is based on the effect on the Boolean function of the circuit. Any logic circuit can be represented by a directed graph, where the gates, primary inputs and outputs are represented by vertices of the graph and the lines between gates are represented by the arcs of the graph. Such a graph that describes the structure of the circuit is called as logical model of the circuit [31]. The R structural equivalence and functional equivalence proposed by McCluskey and Clegg [64] are summarized here [62].

R Structural Equivalence: Two faults f_1 and f_2 are said to be R structurally equivalent (written $f_1 \cong f_2$) if they produce the same reduced circuit graph when faulty values are implied and constant edges are removed.

Functional Equivalence: Two faults f_1 and f_2 are said to be functionally equivalent (written $f_1 \simeq f_2$) if they modify the Boolean function of the circuit in the same way, i.e., they yield the same output function.

It should be noted that structural equivalence implies functional equivalence. Hence, using functional equivalence collapsing we get a smaller set than with structural equivalence collapsing. However, functionally equivalent faults, which are not structurally equivalent, are caused by the presence of reconvergent fanout paths in the network [64]. Functional collapsing is explained in greater detail in Chapter 3.

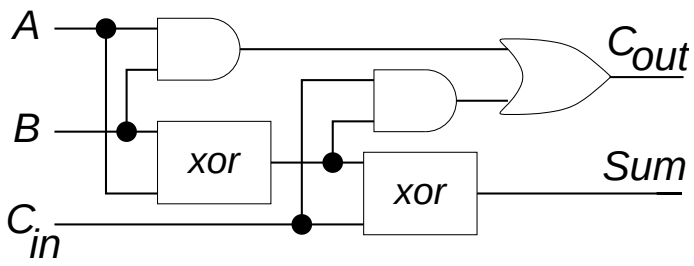

Figure 2.1: Full adder circuit.

Structural fault collapsing alone can reduce the fault set size to about 50% of all faults [24]. Most ATPG programs use only structural equivalence fault collapsing. The Fastest program [56], developed at the University of Wisconsin, can do both equivalence and dominance collapsing, but it does only structural collapsing. The relative size of the collapsed set with respect to the size of all faults is called the collapse ratio [24]:

$$Collapse\ ratio = \frac{|Set\ of\ collapsed\ faults|}{|Set\ of\ all\ faults|} \tag{2.1}$$

As an example, consider the full adder circuit shown in Figure 2.1 where the XOR blocks are implemented as shown in Figure 2.2. The test generation package Hitec [68] generates a structural equivalence collapsed set of 38 faults (collapse ratio = 0.63) out of the total fault set of 60. The structural dominance collapsing obtained using Fastest [56] results in 30 faults (collapse ratio = 0.5). Using functional collapsing, the fault set can be further reduced as explained in Chapter 3.

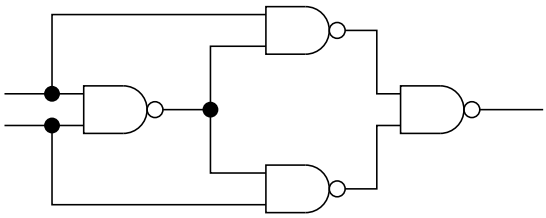

Figure 2.2: XOR function implementation.

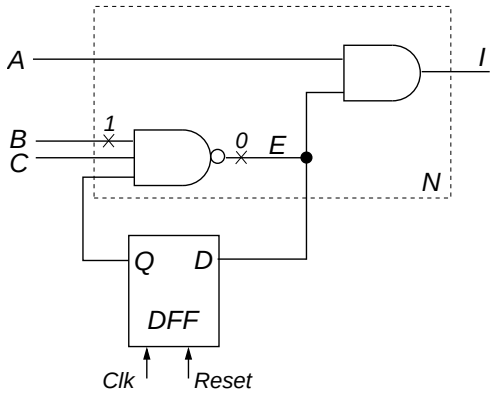

Figure 2.3: A sequential circuit.

2.3 Collapsing in Sequential Circuits

The preceding discussion deals with combinational circuits. The fault collapsing within and across sequential elements is more complex [26, 29]. It has been shown [1, 20] that the dominance relations in a combinational circuit N may not remain valid when N is embedded in a sequential circuit, while the equivalence relations of N stay valid. Such a case has been demonstrated in Figure 2.3, which is taken from the paper by Chen *et al.* [29].

Figure 2.3 shows a sequential circuit with the combinational part, N, enclosed in a dashed box. Considering the combinational part, we would say the s-a-1 fault on line B, say α, is dominated by the fault s-a-0 on line E, say β. The vector sequence $\{000, 001, 111\}$ on inputs A, B, C detects the fault α, but does not detect the fault

β. So, in the sequential circuit, β does not dominate α and this is because of a phenomenon called self hiding as explained in [29].

The paper by Chang and Breuer [26] uses a multiple-fault checkpoint-labeling procedure while the paper by Chen *et al.* [29] gives a single fault collapsing analysis. The latter introduces two phenomena, self hiding and delayed reconvergence, which invalidate the dominance relations of the combinational part of the circuit. It also explains the equivalence relationship between the faults at some specific fanout branches and their corresponding fanout stems. New fault relationships in D flip-flops are also identified to further collapse faults at feedback lines. The paper by Boppana and Fuchs [18] does the dynamic fault collapsing using a test pattern generator, while Chen *et al.* [29] do the collapsing statistically, that is, before any test pattern generation. Table 2.1 shows the dominance collapsed results obtained by using the techniques explained in [29] for a few ISCAS'89 benchmark circuits [21]. These values are compared with the values under column Comb. which are obtained by a dominance collapsing procedure typically used for combinational circuits [56]. In Table 2.1, the collapsed set size under column Chen [29] is smaller for two circuits and greater for the other two than the set size under column Comb. [56]. The increase in the collapsed set size is because of the dominance relations that are invalidated due to phenomena like self hiding and delayed reconvergence introduced by Chen *et al.* [29]. New fault relationships across D flip-flops and fanouts that Chen *et al.* described in [29] cause a decrease in the collapsed set size.

Table 2.1: Dominance collapsing results for ISCAS'89 circuits.

Circuit	All Faults	Comb. [56]	Chen [29]
s27	52	25	16
s344	670	265	316
s1196	2392	942	928
s9234	18468	5505	6522

2.4 Fault Sampling

In circuits with very large number of faults, fault set size reduction beyond that obtained by fault equivalence or fault dominance collapsing may be needed for producing a set of target faults of reasonable size. Fault sampling can be used in this case to reduce the size of the set of target faults [4, 7, 13]. However, unlike fault collapsing, fault sampling can not provide a single set of faults whose detection guarantees the detection of all the detectable faults [73].

2.5 Graph Model

The graph model described here is the same as given in [11, 75]. The fault equivalence and dominance relations are represented by a directed graph. In this graph each fault is represented by a node. If fault f_2 dominates fault f_1, then this is represented by a directed edge from the node representing f_1 to the node representing f_2. This edge indicates that any test for f_1 must detect f_2. Clearly, the presence of edges $f_1 \rightarrow f_2$ and $f_2 \rightarrow f_1$ indicates that the two faults f_1 and f_2 are equivalent. A fault dominance graph, or simply a dominance graph, represents the dominance relations among the faults of a circuit.

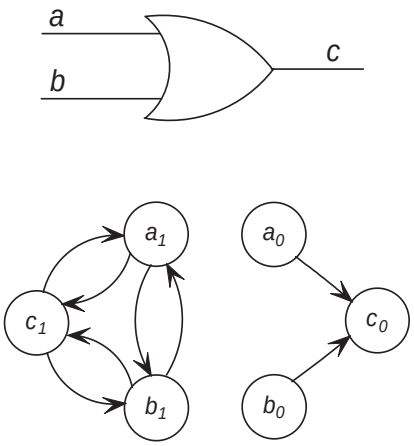

Figure 2.4: Dominance graph of an OR gate.

Table 2.2: Dominance matrix of an OR gate.

	a_0	a_1	b_0	b_1	c_0	c_1
a_0	1	0	0	0	1	0
a_1	0	1	0	1	0	1
b_0	0	0	1	0	1	0
b_1	0	1	0	1	0	1
c_0	0	0	0	0	1	0
c_1	0	1	0	1	0	1

Figure 2.4 shows the dominance graph for all faults of an OR gate. The subscript fault notation has been used, that is, a_0 means that the fault is on line named 'a' and is s-a-0 type. The dominance graph is conveniently represented by its connectivity matrix, specifically called dominance matrix, as shown in Table 2.2. A 1 entry at the intersection of a row and a column means that the fault corresponding to the column dominates the fault corresponding to the row. For example, the 1 in the second row and the last column indicates that c_1 dominates a_1. Equivalence of two faults is expressed by two 1's placed at the intersections of the rows and columns corresponding to those faults. Since there is also a 1 in the last row and second column, indicating that fault a_1 dominates fault c_1, it can be said that a_1 and c_1 are

equivalent. An equivalence is indicated by 1's that are placed in diagonally symmetric positions. Dominance alone is indicated by an asymmetric 1. This dominance matrix is used in algorithms of Chapters 4 and 5 to represent all the dominance relations between the faults.

Chapter 3

Prior Work on Functional and Hierarchical Fault Collapsing

In this chapter, the background work on functional and hierarchical fault collapsing is presented.

3.1 Prior Work on Functional Collapsing

It has been mentioned in the previous chapter that functional collapsing results in a smaller collapsed set than structural collapsing. But functional collapsing is more expensive as it depends on the output function of the circuit. It can be shown [43, 53] that identifying fault equivalence between two arbitrary faults in a combinational circuit is an NP-complete problem because it requires proving the equivalence of the two faulty functions. There are other methods [44, 63, 67, 83] which are not of exponential complexity but they can find only some, not all, functional equivalences. Goundan and Hayes [44] propose a technique to determine equivalences, if any, between faults on primary input and outputs. A technique by Nadjarbashi *et al.* [67] finds equivalence relationships between faults on reconvergent fanout stems and those on reconverging lines. Vaaje [83] gives a mathematical approach to detect such equivalences. The simplest of all fault collapsing methods is to use the checkpoints of the circuit [24]:

Checkpoints: Primary inputs and fanout branches of a combinational circuit consisting only of Boolean gates are called checkpoints.

Checkpoint Theorem: A test set that detects all single stuck-at faults of the checkpoints of a combinational circuit detects all single stuck-at faults in that circuit.

It has been proven by Chen *et al.* [28] that the primary inputs that fanout need not be checkpoints for the class of irredundant two-level combinational circuits. In their earlier paper [27], they proved that for a general circuit, all primary inputs have to be considered as checkpoints.

There is a problem with using checkpoint sets. If the test set has a less than 100% fault coverage, that is, for circuits with redundant faults, the checkpoint theorem is not guaranteed [2]. Thus, in general, the set of checkpoint faults does not provide a sufficient set of target faults. The inadequacy of checkpoint faults stems from the fact that checkpoint theorem is based on dominance collapsing. Abramovici *et al.* [2] provide a procedure to select the additional target faults so as to ensure sufficiency. Later, that procedure was critically reviewed and modified by Gopalakrishnan and Bhattacharya [42].

The above discussion on checkpoints deals with single stuck-at faults for combinational circuits. For multiple stuck-at faults in combinational circuits, Bossen and Hong [19] have derived the following checkpoint labeling procedure:

- All the primary inputs that do not fanout are checkpoints.
- All the fanout branches are checkpoints.
- NOT gates are considered as lines in this procedure.

A primary input that is also a fanout stem is considered as a checkpoint when dealing with single stuck-at faults, but is not considered for multiple stuck-at faults. For discussion on checkpoint labeling procedure for sequential circuits, the reader is referred to a paper by Chang and Breuer [26].

3.1.1 Functional Equivalence

For an input vector, V, to be a test for a fault, we have [8]

$$F_0(V) \oplus F_1(V) = 1 \tag{3.1}$$

where F_0 is the fault-free function and F_1 is the faulty function of the fault. Consider a second fault that produces a faulty function F_2. According to the definition of fault equivalence, two equivalent faults have exactly the same tests. Therefore, for two faults to be equivalent, we have

$$[F_0(V) \oplus F_1(V)] \oplus [F_0(V) \oplus F_2(V)] = 0 \tag{3.2}$$

Manipulation of Equation 3.2 results in the following equation:

$$F_1(V) \oplus F_2(V) = 0 \tag{3.3}$$

which means that the two faulty functions are identical for all input vectors. This is considered as a general definition for functional equivalence. These equations are functionally depicted in Figure 3.1. If the fault in block F_1 is equivalent to the fault in

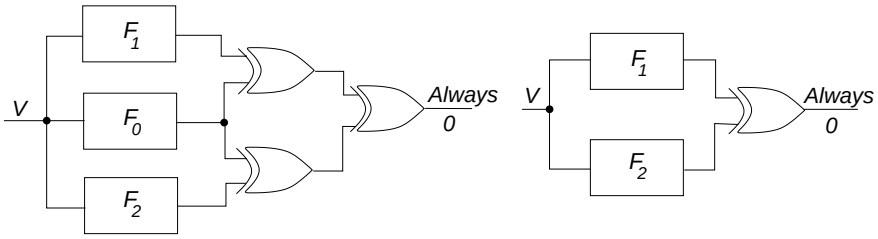

Figure 3.1: Two ways to view fault equivalence.

block F_2, then the outputs of the circuits in Figure 3.1 are always zero. Considering the full adder example shown in Figure 2.1, the functional equivalence collapsing leads to 26 faults, a collapse ratio of 0.43. In comparison, the structural equivalence collapsed set for this circuit has 38 faults and a collapse ratio of 0.63.

The papers by Lioy [61, 62] propose functional fault collapsing based on D-frontiers used in the automatic test pattern generation (ATPG) procedures of Roth *et al.* [77]. There are also other works to find fault equivalences using ATPG [45, 84] and simulation [14]. Fault equivalence identification can be based on redundancy information [15], and hence test generation can prove equivalence [47]. All these techniques have high complexity and so they cannot be used for large circuits. Al-Asaad and Lee [14] propose a simulation-based approach to find global equivalences in large circuits. Since their method is approximate, it may fail to find some equivalences.

3.1.2 Functional Dominance

If a fault f_1, with faulty function F_1, dominates another fault f_2, with faulty function F_2, then the two faults are functionally equivalent for the input vector set that tests fault f_2, i.e., all tests of f_2 satisfy Equation 3.3 [1]. Let vector V detect f_2,

so it must satisfy the following equation:

$$F_0(V) \oplus F_2(V) = 1 \tag{3.4}$$

Since f_1 dominates f_2, any vector that satisfies Equation 3.4 must satisfy Equation 3.1. Also, by contrapositive law, any vector that does not satisfy Equation 3.1 must not satisfy Equation 3.4. These conditions are combined in Equation 3.5 that must be satisfied by all input vectors.

$$[F_0(V) \oplus F_2(V)][\overline{F_0(V) \oplus F_1(V)}] = 0 \tag{3.5}$$

This relation, as shown in Figure 3.2, was explained in the paper by Agrawal *et al.* [8]. Equation 3.5 reduces to:

$$\overline{F_1(V)}F_2(V)\overline{F_0(V)} + F_1(V)\overline{F_2(V)}F_0(V) = 0 \tag{3.6}$$

Although it is not obvious, this condition is consistent with the D-frontier representation of functional fault dominance given by Lioy [61, 62]. If the fault present in block F_1 dominates the fault present in block F_2, then the output of the circuit in Figure 3.2 is always zero.

For the full adder shown in Figure 2.1, the functional dominance collapsed set size is 12, giving a collapse ratio of 0.2.

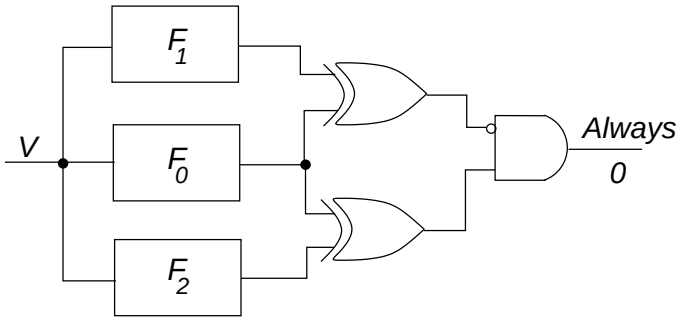

Figure 3.2: Viewing fault dominance.

3.1.3 Level of Similarity

Pomeranz and Reddy [73] recently described another approach to fault collapsing based on a metric called level of similarity between faults. A fault f_j is said to be similar to a fault f_i with a level of similarity $SL_{i,j} \leq 1$ if a fraction, $SL_{i,j}$, of the tests for f_i also detect f_j. If $SL_{i,j}$ is high enough, one may exclude f_j from the set of target faults and rely on the test for f_i to detect f_j as well. When fault f_j is equivalent to f_i or when f_j dominates f_i, then level of similarity $SL_{i,j} = 1$. They obtained collapsed sets of faults that are around 30% of the conventional collapsed sets, without compromising on the fault coverage, for the combinational parts of the benchmark circuits.

3.1.4 Redundant and Aborted Faults

A redundant fault, as explained in Section 2.1, is a fault that does not have any test vector. According to the definition of dominance, a redundant fault is dominated by any other fault in the circuit. This is confirmed by the scheme of Figure 3.2 as a redundant fault introduced in block F_2 will cause a 0 at the output of the bottom XOR gate and will always lead to a 0 at the output. Any two redundant faults dominate

each other and are functionally equivalent; neither fault modifies the functionality of the circuit.

A deterministic test generator, as explained in Section 2.1, tries to find a test vector for a given target fault. Since the search process is NP-complete [37, 53], a limit is generally imposed on the CPU time or the number of backtracks in the search. When the limit is reached and the test generator is not able to decide whether the fault is detectable or redundant, the fault is called an aborted fault [55]. An aborted fault has an unknown status, but in reality, it can be either redundant or detectable. If a fault is aborted in a circuit, then the detectability of this fault when introduced in either of the blocks F_1 or F_2 in Figures 3.1 and 3.2 is also unknown. So, aborted faults have to be considered separately. This will be discussed in Section 4.2.2.

3.2 Prior Work on Hierarchical Fault Collapsing

As explained earlier, functional collapsing is not recommended for large circuits. So, most ATPG programs [30, 56, 59, 68] use only structural collapsing. Hierarchical fault collapsing seems to be an alternative that allows some functional collapsing. In this section, we present a detailed overview of two earlier techniques [46, 75] used for hierarchical fault collapsing. Both techniques result in a collapse ratio lower than that obtained by structural collapsing alone, but not as low as obtainable if a complete functional collapsing could be used.

The increasing complexity of the circuits can be efficiently handled using a hierarchical design process. The hierarchical description of a circuit at the highest level consists of an interconnection of few blocks, which are described at a lower level of

hierarchy consisting of other blocks and gates [86]. A block or a sub-circuit C_j in a circuit C_i is called an instance of C_j. The circuit defined by hierarchical description can be obtained by an expansion process referred to as flattening.

The hierarchy of a circuit description can be used in test generation [60, 85, 87] and fault simulation [58, 66, 76]. To the best of the author's knowledge, there have been few papers [8, 9, 46, 75] that have used hierarchy of the circuit to collapse faults. There are advantages of using hierarchy to collapse faults. A reduced fault set, that is computed once for a sub-circuit, can be reused for all instances of the sub-circuit. For smaller circuits, functional fault collapsing techniques can be used so as to achieve better collapse ratios.

3.2.1 Prasad *et al.*'s Method

The paper by Prasad *et al.* [75] presents a graph-theoretic algorithm for collapsing faults. Functional equivalences for logic cells in a library-based design are pre-computed using the construction of Figure 3.1 and saved for use in the hierarchical collapsing procedure. The transitive nature of the dominance relationship is used, i.e., if a fault f_1 dominates fault f_2, which in turn dominates fault f_3, then fault f_1 also dominates f_3. These relations can be obtained by representing the fault dominances in a graph, called a dominance graph, and then computing the transitive closure of the dominance graph to get the dominance matrix. There are efficient algorithms of computing transitive closure [10, 34], but this paper uses the Floyd-Warshall

algorithm, which is of complexity $O(n^3)$ [32]. Then *algorithm equivalence* and *algorithm dominance* are executed to obtain the equivalence and dominance collapsed set. These algorithms, as repeated here, will be used later in Chapter 4.

Algorithm Equivalence:

Begin with F (set of all faults) and E (equivalent set, initially empty). Execute the following steps until F becomes empty:

1. Arbitrarily select a fault from F.
2. Intersect (bit-by-bit logical AND) the row and column vectors of the dominance matrix corresponding to the selected fault.
3. The set of faults corresponding to 1's in the intersected vector is the equivalent set. Add any one fault from this set to E and delete all faults of this set from F.

Algorithm Dominance:

Begin with E (equivalent collapsed set obtained from *algorithm equivalence*). Execute the following steps:

1. Reduce the dominance matrix by deleting the rows and columns corresponding to all faults that are not in E.
2. Remove from E the faults whose columns in the reduced dominance matrix have any off-diagonal 1's. The remaining set E is the dominance collapsed fault set.

For the full adder, implemented as shown in Figure 2.1, the dominance matrix is obtained using all the functional equivalences present in the XOR block of Figure 2.2. When *algorithm equivalence* and *algorithm dominance* were used on this matrix, it resulted in hierarchical equivalence and dominance collapsed sets of 30 and 24, respectively. These numbers were later corrected in their subsequent paper [8] as 26 and 20. These values are better than the structural equivalence and dominance collapsed sets of 38 and 30. Prasad *et al.* [75] use functional equivalences for the sub-circuits, while Agrawal *et al.* [8] use functional dominance along with hierarchy. The latter method gives a dominance collapsed set of 14 (collapse ratio of 0.23) for the full adder circuit. It was demonstrated by Agrawal *et al.* [9] that in general, using hierarchy, the dominance collapsing may result in a collapse ratio lower than 25%.

3.2.2 Hahn *et al.*'s Method

The paper by Hahn *et al.* [46] also does equivalence collapsing using the hierarchy of the circuit. This paper does an exact equivalence analysis based on reduced ordered binary decision diagrams [23], which allows the construction of canonical representations of Boolean functions.

The subscript fault notation explained in Section 2.5 is inadequate, because the lines which exist in the hierarchical description of the circuit may no longer exist in the non-hierarchical (flattened) description. The input and output lines of the sub-circuit are deleted in the expanded version. Therefore, it is necessary to identify a fault in terms of the 'real gates'. One such fault notation is used by the test generation program Gentest [30], which can handle hierarchical designs. A fault is specified as

GateA(GateB, b), where b can be a 0 or 1 and defines the type of fault. It means that the fault is on the input line of the gate *GateA* coming from gate *GateB*. *GateA* can be the name of a gate or a primary input of the circuit. *GateB* is the name of a gate. *GateB* is optional and a fault *GateA(b)* means a stuck-at-b on the output of gate or on a primary input named *GateA*. For a 2-level deep hierarchical circuit, a fault in the sub-circuit is specified as *GateC.GateA(GateB, b)*, where *GateC* is the instance name of the sub-circuit at the top level and *GateA* and *GateB* are the names in the sub-circuit. This way, faults at any level can be represented using a "period" to distinguish the levels.

The collapsed fault set of a sub-circuit can be reused for all instances of the sub-circuit. However, problems arise when different embeddings of the sub-circuit are considered. For faults located on the inputs and outputs of sub-circuit, it depends on the embedding whether or not the fault has to be considered. This problem has been solved in this paper using two different sets of faults for each sub-circuit. One set contains all faults that have to be considered independently of the embedding, and the other set depends on the embedding of the sub-circuit. Algorithms HFSG (Hierarchical Fault Set Generation) and HFSG$_I$ (Improved HFSG) are presented in the paper which collapse the faults in a hierarchical circuit. It has been shown that, on an average, the reduction in the collapsed set size is 23%.

This technique can be used even for sequential circuits, though they do not incorporate any of the techniques proposed in [29]. The work by Hahn *et al.* deals only with equivalence collapsing, while Prasad *et al.* find both equivalence and dominance collapsed sets.

Table 3.1: Collapsing for a full adder circuit.

	Structural [56, 68]	Hierarchical [8, 9, 75]	Functional*
Equivalence	38 (0.63)	26 (0.43)	26 (0.43)
Dominance	30 (0.50)	14 (0.23)	12 (0.20)

3.3 Summary

In this chapter, we have explained the various techniques used for functional and hierarchical fault collapsing. The circuit of full adder, where the XOR block is implemented using four NAND gates, as shown in Figures 2.1 and 2.2, is used as an example circuit in all cases. The summary of the collapsed set sizes using different collapsing techniques is tabulated in Table 3.1. The collapse ratios, as given by Equation 2.1, are shown by the accompanying fraction in parenthesis. The values under the column Functional* are obtained when functional collapsing techniques described in this chapter are used. It can be seen from Table 3.1 that the collapse ratios obtained with functional collapsing are better than hierarchical collapsing, which in turn, are better than structural collapsing.

Chapter 4

Functional Collapsing

In this chapter, the definitions of equivalence and dominance are clarified, especially for the case of multiple output circuits. Consider the circuit in Figure 4.1, which has two outputs. As explained in Section 2.1, a typical test generation process does fault simulation after a test is derived for some fault. Consider the fault $Y(0)$, which is detected by the test vector 11 on the inputs A and B. On performing fault simulation, the fault $Z(B,\ 0)$, detectable by the same vector, is dropped. Had we started with fault $Z(B,\ 0)$, then the fault simulation would have dropped $Y(0)$. Both faults have the same test vector, so whenever one is detected, the other is dropped through fault simulation. But these faults are not considered equivalent according to the conventional definition of equivalence. This happens only for circuits with more than one output. This observation motivated the work presented in this chapter.

4.1 Definitions

In this section, we first quote the conventional definitions of equivalence and dominance and then extend them for multiple output circuits.

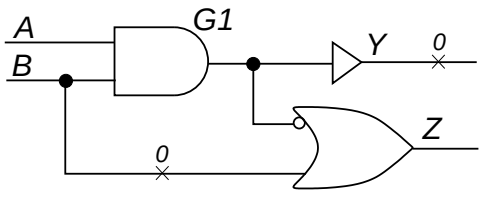

Figure 4.1: A circuit with two outputs.

Definition 1: *Fault Equivalence* [1, 24] - Two faults are equivalent if and only if the corresponding faulty circuits have identical output functions.

Definition 2: *Fault Dominance* [55, 69] - A fault f_i is said to dominate fault f_j if (a) the set of all vectors that detects fault f_j is a subset of all vectors that detects fault f_i and (b) each vector that detects f_j implies identical values at the corresponding outputs of faulty versions of the circuit.

These definitions are extended for possible interpretations for multiple output circuits.

Definition 3: *Diagnostic equivalence* - Two faults of a Boolean circuit are called diagnostically equivalent if and only if the functions of the two faulty circuits are identical at each output.

This definition of equivalence implies indistinguishability of two faults, i.e., the effects of the two faults can not be distinguished at primary outputs of the circuit.

Definition 4: *Detection equivalence* - Two faults are called detection equivalent if and only if all tests that detect one fault also detect the other fault, not necessarily at the same output.

Two detectable faults that are diagnostic equivalent are also detection equivalent, but the reverse implication is not true. Faults that are detection equivalent need not be indistinguishable. The faults shown in Figure 4.1 are detection equivalent. These faults are distinguishable as they produce distinguishing patterns at primary outputs. Inherently, the fault simulation uses detection relations, while it is often incorrectly believed to be functional (diagnostic) relations.

Definition 5: *Diagnostic dominance* - If all tests of a fault f_1 detect another fault f_2 on the exact same outputs where f_1 was detected, then f_2 is said to diagnostically dominate f_1.

Definition 6: *Detection dominance* - If all tests of a fault f_1 detect another fault f_2, irrespective of the output where f_1 was detected, then f_2 is said to detection dominate f_1.

The detection dominance is same as the *test covering* relation proposed by Abramovici *et al.* [3] or *test implication* proposed by To [81]. The detection equivalence is referred to as *test equivalence* by Lioy [63] and To [81]. Like equivalence, diagnostic dominance between two detectable faults also implies detection dominance.

We note that Definitions 3 and 5 (diagnostic type) are identical to Definitions 1 and 2 that are discussed in the literature. Definition 2 has been referred to as *column dominance* by Poage [69]. This definition was later modified as, "a fault f_i is said to dominate fault f_j if the faults are equivalent with respect to the test set of fault f_j" [1, 24]. This definition means exactly the same as Definition 2, if equivalence is interpreted as diagnostic equivalence. Sometimes, the same definition is modified as "if all tests of fault f_1 detect another fault f_2, then f_2 is said to dominate f_1" [24]. This definition is referred to as *column dominance* by Schertz and Metze [79]. It is the same as part (a) of Definition 2 without part (b). This variation of Definition 2 implies detection dominance, whereas Definition 2 means diagnostic dominance.

A definition of dominance for sequential circuits appears in the paper by Poage and McCluskey [70]. Goel [40] has studied fault dominance among single and multiple stuck-at faults in sequential circuits.

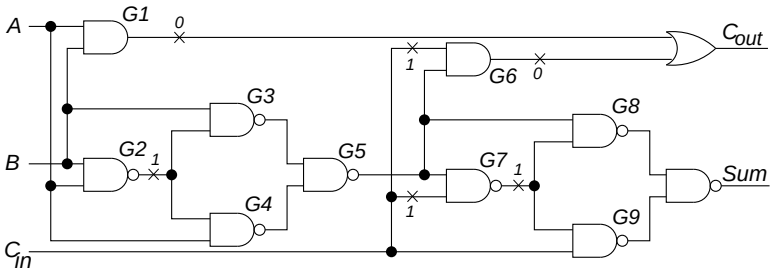

Figure 4.2: Full adder circuit.

Two faults that dominate each other are equivalent. The same conclusion is applicable for diagnostic and detection relations too. Any two faults that detection dominate each other are detection equivalent. For circuits with only one output, diagnostic definitions are the same as detection definitions. It should be noted that two redundant faults (implying redundancy at all primary outputs) are equivalent according to any of the equivalence definitions (Definitions 1, 3 and 4).

The example of the full adder that was considered in earlier chapters is shown here in Figure 4.2, but with the XOR blocks replaced with four NAND gate implementations. For this circuit, diagnostic collapsing results in collapsed set sizes of 26 and 12 for equivalence and dominance, respectively. Using detection collapsing, the fault set sizes are 23 and 6 for equivalence and dominance, respectively. A collapsed set of 6 faults for detection dominance is the least of all the reported results. There are three faults that are found to be detection equivalent with other faults in the diagnostic equivalent fault set. These faults, in Gentest fault notation detailed in Section 3.2.2, are $G1(0)$ and $G2(1)$, $G6(C_{in},\ 1)$ and $G7(C_{in},\ 1)$, and $G6(0)$ and $G7(1)$, as shown in Figure 4.2.

In Figure 4.2, the fault $G1(0)$, which is detected at the output line C_{out} with the vector 11x on the inputs A, B, C_{in}, is considered as detection equivalent with

$G2(1)$, according to Definition 4. But the same vector 11x does not detect the fault $G2(1)$ at either of the outputs C_{out} and Sum, while the individual vectors 110 and 111 detect the fault at the output Sum. Such relations, based on incompletely specified input vectors, are addressed by Pomeranz and Reddy [74]. According to them, the detection equivalent faults $G1(0)$ and $G2(1)$ are not considered as $0, 1, x-equivalent$, but are treated as $0, 1-equivalent$. The equivalence of two faults over an incompletely specified test set is referred to as $0, 1, x-equivalence$. The functional equivalence according to Definitions 1 and 3 can be seen as $0, 1, x-equivalence$, while Definition 4 can be seen as $0, 1-equivalence$. The dominance definitions (Definitions 5 and 6) quoted above are over the test set of {0,1} and, not over {0,1,x}.

4.2 Functional Dominance

Finding functional dominances using the construction of Figure 3.2 is a computationally expensive procedure. This is because we need to implement it for all permutations of faults taken two at a time. A modified and less expensive scheme is shown in Figure 4.3 where, initially all three blocks are fault free copies of the circuit with function F_0. Consider a fault, say f_1, and introduce it in the bottom block whose function is now designated as F_1. Consider another fault, say f_2, which is dominated by f_1 in the given circuit. Whenever f_2 in the top block is activated and propagated to the AND gate, it is blocked by the output of the XOR gate (a logic 1), because fault f_1 is also detectable when f_2 is detected. So, all faults that are dominated by f_1 in the given circuit are redundant in the top block. In a single iteration of the ATPG, we will find all faults that are dominated by the fault introduced in block F_1.

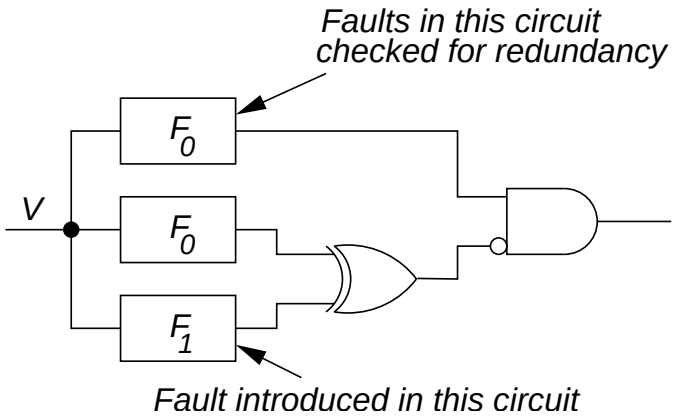

Figure 4.3: Identifying functional dominance.

4.2.1 Algorithm

1. Select an unprocessed fault of the given circuit and build a circuit as shown in Figure 4.3 with the fault introduced in the bottom block whose function is F_1.

2. Check for redundant faults in the top block F_0.

3. For each redundant fault found in step 2, a 1 is placed in the dominance matrix at the intersection of the row corresponding to the redundant fault and the column corresponding to the fault in the bottom block. Thus, we obtain all values of a column of the dominance matrix in a single iteration.

4. Go to step 1 until there is no fault left.

5. Now, we will have the dominance matrix with all functional dominance relations included.

6. Transitive closure of the dominance matrix is computed, which is then reduced using the *algorithm equivalence* of Prasad *et al.* [75]. This reduced matrix consists of the dominance relations within an equivalence collapsed set of faults.

7. If dominance collapsing is required, then the reduced matrix of the previous step is further reduced according to *algorithm dominance* of Prasad *et al.* [75].

As pointed out in Section 3.1.4, the redundant and aborted faults of the given circuit (stand-alone F_0) have to processed differently, which is explained in the next section.

4.2.2 Redundant and Aborted Faults

A redundant fault of the given circuit will be redundant in the top block, F_0, of Figure 4.3 for any fault introduced in the bottom block F_1. This will cause a row of all 1's corresponding to all redundant faults of the given circuit in the dominance matrix. After *algorithm equivalence*, all redundant faults are collapsed into one fault. This row should be removed before *algorithm dominance*. Otherwise, *algorithm dominance* will remove all detectable faults leaving just the redundant fault, because a redundant fault is considered to be dominated by any other fault.

There can be some aborted faults while checking for redundancies in step 3 of the algorithm in Section 4.2.1. Aborted faults, as discussed in Section 3.1.4, have unknown status and can be redundant or detectable. Had these aborted faults been treated as redundant we would have inserted additional 1's in the dominance matrix resulting in a smaller, though possibly erroneous, collapsed set. This is the very reason why the transitive closure is computed in step 6 of the algorithm so that some relations that were missed because of aborted faults are retained. Though this helps, it can not guarantee to find all the escaped dominance relations. However, for the cases where no fault is aborted, transitive closure is not needed.

When the given circuit is subjected to test generation, any fault can be detected, aborted or declared as redundant. For each of these cases, Table 4.1 gives the value that should be entered in the dominance matrix in step 3 of the algorithm in Section 4.2.1. The first column corresponds to the three cases of detectability of the fault in stand-alone F_0. The cases corresponding to the detectability of the same fault in the top block, F_0, of Figure 4.3 are shown in the second column. When a detectable fault of the given circuit is shown as aborted in the top block of Figure 4.3, we place a 0 in the dominance matrix, which means that the aborted fault is considered as detectable for reasons as explained in the previous paragraph. When a redundant fault of the given circuit is shown as aborted in Figure 4.3, then we introduce a 1 in the dominance matrix. In this case, we are considering the aborted fault as redundant. This is a correct decision, because a redundant fault in the given circuit will also be redundant in the top block of Figure 4.3. If an aborted fault in the given circuit is seen as aborted in the top block of Figure 4.3, we introduce an x in the dominance matrix. At the end of step 5 of the algorithm of Section 4.2.1, we examine the rows with x's in the dominance matrix. If there is at least one 0 in the row, then all the x's of that row are replaced with 0's. In this case, the aborted fault in the given circuit is detectable at least once in the top block, F_0, of Figure 4.3. Hence, the aborted fault is not a redundant fault in the given circuit. So, we treat the aborted fault as a detectable fault and replace the x's of its row by 0's. If the row with x's had only 1's other than x's, then the x's are replaced with 1's. If the aborted fault was never detected in the top block of Figure 4.3 and shown as redundant at least once, then we consider the aborted fault as redundant fault by replacing the x's by 1's. Such a

Table 4.1: The dominance matrix entry.

Detectability of fault in stand-alone F_0	Detectability of fault in top block, F_0 of Figure 4.3	Value in dominance matrix
Detectable	Detectable	0
	Redundant	1
	Aborted	0
Redundant	Redundant	1
	Aborted	1
Aborted	Detectable	0
	Redundant	1
	Aborted	x

row with all 1's is removed before *algorithm dominance*. If the row had only x's, then all the x's are replaced with 0's. Then, this fault will be considered in the dominance collapsed set.

Since, a redundant fault of the given circuit has only 1's in the corresponding row of the dominance matrix, which is anyway removed before *algorithm dominance*, we can consider only non-redundant faults with the algorithm. In the implementation of Figure 3.2, the ATPG is run $n(n-1)$ times, n being the number of non-redundant faults in the circuit, and in each run of ATPG, we test the redundancy of one fault. Using the scheme in Figure 4.3, the ATPG is run n times and in each run, we carry out the redundancy test for $n-1$ faults. In the implementation of Figure 4.3, the circuit is built and prepared n times, which is less than the $n(n-1)$ times needed for the implementation of Figure 3.2.

The scheme of Figure 4.3 is for a single-output circuit. For a circuit with multiple outputs, the schemes of Figure 4.4 are used. It illustrates a case of two outputs and the generalization for more than two outputs is straightforward. The scheme of Figure 4.4(a) conforms to the diagnostic collapsing as given by Definitions 3 and 5 of

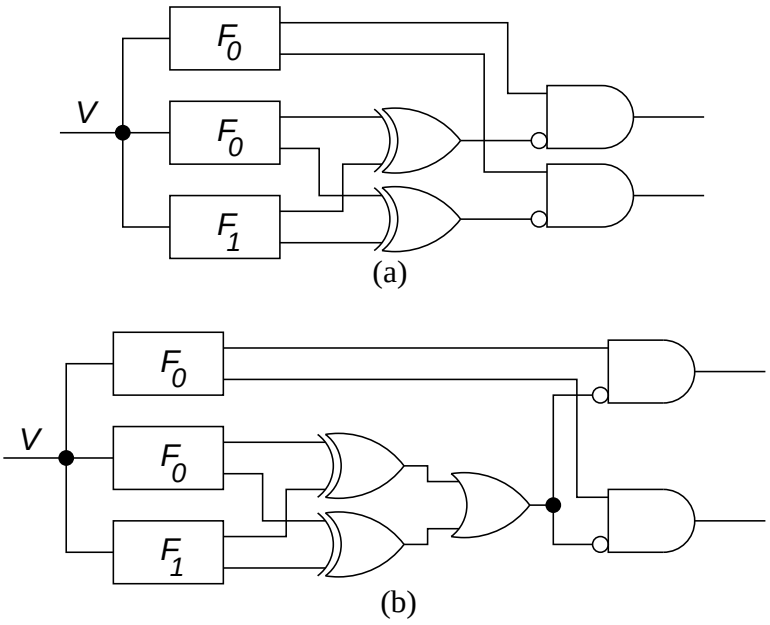

Figure 4.4: Schemes for collapsing faults with a) diagnostic and b) detection criteria.

Section 4.1, while that of Figure 4.4(b) does detection collapsing. It should be noted that the scheme of Figure 4.4(b) ensures that a test vector which detects the fault introduced in the block F_1 on an output line blocks all output AND gates because of the additional OR gate, unlike in Figure 4.4(a). This causes more redundancies and therefore more dominance relations between faults, that is, more 1's in the dominance matrix. Hence, the fault set obtained using detection dominance will be smaller than that obtained for diagnostic dominance for multiple output circuits.

4.3 Results

We present four example circuits for illustration. The smallest is a simple XOR function implemented with four NAND gates as shown in Figure 2.2 and the largest is the 4-bit ALU circuit (74181). The 8-bit adder is a ripple carry adder as shown

Table 4.2: Comparison of fault collapsing results.

Circuit name	All faults	Number of collapsed faults (Collapse Ratio)							
		Structural [56, 68]		Functional [8]		Functional collapsing - New results			
						Diagnostic		Detection	
		Equiv.	Dom.	Equiv.	Dom.	Equiv.	Dom.	Equiv.	Dom.
XOR	24	16 (0.67)	13 (0.54)	10 (0.42)	4 (0.17)	10 (0.42)	4 (0.17)	10 (0.42)	4 (0.17)
Full adder	60	38 (0.63)	30 (0.50)	26 (0.43)	14 (0.23)	26 (0.43)	12 (0.20)	23 (0.38)	6 (0.10)
8-bit adder	466	290 (0.62)	226 (0.49)	194 (0.42)	112 (0.24)	194 (0.42)	96 (0.21)	191 (0.41)	48 (0.10)
ALU (74181)	502	301 (0.60)	248 (0.49)	–	–	253 (0.50)	155 (0.31)	234 (0.47)	92 (0.18)

in [75]. The results according to the definitions of diagnostic and detection collapsing of Section 4.1 are tabulated in Table 4.2. These results are compared with the structural equivalence collapsed set obtained using either Hitec [68] or Fastest [56] programs, and the structural dominance collapsed set obtained using the Fastest [56] program. The column "Functional [8]" has the values obtained using a hierarchical fault collapsing technique [8]. The XOR gates of the ALU circuit are replaced with the four NAND gate implementation as shown in Figure 2.2. The collapse ratio, as defined by Equation 2.1, of each collapsed set is shown by the accompanying fraction in parenthesis.

The functional equivalence set for the XOR gate results in 10 faults. It is proved by Goundan and Hayes [44] that every irredundant realization of a two-variable Exclusive-OR function has a unique set of ten fault classes. For the 8-bit adder, the previous best result is reported by Agrawal *et al.* [8]. Their hierarchical collapsing technique resulted in the dominance collapsed set of 112 faults (collapse ratio 0.24) while the implementation described in this chapter leads to a set of 48 faults

(collapse ratio 0.10), a reduction of over 50%. This result is for detection dominance. The diagnostic dominance results in a collapse ratio of 0.21, which still is smaller than that previously reported [8]. The dominances that are missed by the hierarchical collapsing technique of [8] are the functional dominances between faults of different full adder cells that cannot be found using the transitive closure of the dominance graph. Based on our experience, the collapsing using the detection dominance leads to collapse ratios in the range of $0.10 - 0.20$. The paper by Agrawal *et al.* [9] achieves a collapse ratio of about 25%, which is based on diagnostic dominance.

The ATPG used for collapsing algorithms is Hitec/Proofs [68]. There were some aborted faults while checking for redundancies in step 4 of the algorithm in Section 4.2.1. The transitive closure in step 6 of the algorithm is computed so that some relations that were missed because of the aborted faults are retained, as discussed in Section 4.2.2. The number of collapsed faults obtained with detection equivalence for the 8-bit adder using the algorithm was 193. If the computation of transitive closure was not included, the number of faults would have been 194. It has been found that the right value, theoretically using Definition 4 of Section 4.1, is 191. The value reported in our paper [78] should be corrected from 194 to 191. Similarly the detection dominance collapsed set size according to the algorithm was 56. This was later determined as wrong because of the aborted faults and corrected to 48 after manually examining the collapsed set. It is observed that the detection dominance collapsing of a one-bit full adder results in 6 faults and for an n-bit ripple carry adder, it is $6n$ collapsed faults. Similar generalizations for an n-bit ripple carry adder are possible for diagnostic and detection equivalence collapsed sets. The program, when used for

the ALU circuit, also resulted in few aborted faults, so the detection collapsed sets shown against ALU circuit in Table 4.2 need not be the minimum. When no fault is aborted, using a better ATPG, the algorithm would result in the smallest collapsed fault set. But this should not be considered a problem since the use of this algorithm is recommended only for smaller circuits, where we would not expect any aborted faults.

To verify the correctness of the collapsed sets, a test generator was run to derive test vectors for the fault sets obtained from our algorithm. Then a fault simulator is used to determine whether the test vectors detect all faults of the circuit except the redundant and aborted faults. The Gentest ATPG [30] is used for this purpose and the number of test vectors is tabulated in Table 4.3. Gentest is used without any options and this fills the unused inputs of a test vector with previous test vector's steady value on the same input. The test vectors are compared with the test vectors required for the structural equivalence collapsed [68] and dominance collapsed [56] sets. Accompanying each entry of the test vectors, the value in parenthesis is the number of target faults provided to the test generator. The target faults are different from those in Table 4.2 in the case of ALU, because there were 8 redundant faults which are not considered here. Though there is a reduction in the number of test vectors for the ALU, we still have a long way to go because the minimum number of test vectors to detect all the faults is only 12 [12, 48]. It is sometimes observed through experiments that the number of test vectors is dependent on the fault order. We should, however, point out that perhaps a more important factor is the selection of a test vector from among many vectors that detect a target fault.

Table 4.3: Comparison of the test vectors.

Circuit name	Number of test vectors (number of target faults)			
	Structural		Functional	
	Equivalence	Dominance	Diagnostic Dominance	Detection Dominance
Full adder	6 (38)	6 (30)	7 (12)	6 (6)
8-bit adder	33 (290)	28 (226)	32 (96)	28 (48)
ALU	44 (293)	44 (240)	39 (147)	38 (84)

4.4 Summary

In this chapter, the fault equivalence and dominance relations for multiple output combinational circuits are discussed. A novel algorithm based on redundancy identification has been proposed to find the equivalence and dominance collapsed sets corresponding to diagnostic and detection collapsing. The collapse ratios presented are much better (lower) than the earlier known results. The techniques presented to collapse the faults in this chapter use ATPG and their CPU time increases exponentially with the circuit size. These techniques should be used only with smaller circuits and the collapsing results for ALU and 8-bit adder circuits have been included just for demonstration. The collapsing techniques described in this chapter can be used in hierarchical fault collapsing as explained in the next chapter.

The algorithm presented in Section 4.2.1 uses an ATPG to find the dominance of a fault over another fault. The same results can be obtained using a fault simulator instead. The fault simulator is run for all possible input combinations at the primary inputs and correspondingly the relations between faults are obtained according to the definitions in Section 4.1. The time taken with this technique is also exponential in circuit size and so it too can be used only for smaller circuits.

Chapter 5

Hierarchical Fault Collapsing

Typically, faults in a hierarchically described circuit are collapsed after flattening the hierarchy [30]. In our method, we do not flatten the circuit to collapse the faults. Hierarchical fault collapsing is based on the following theorem [44, 46].

Theorem: If two faults are functionally equivalent in a combinational sub-circuit M that is embedded in a circuit N, then they are also functionally equivalent in N.

Functional equivalence here means diagnostic equivalence, as defined in Section 4.1. Such faults are called as intrinsically equivalent in M by Goundan and Hayes [44]. This is demonstrated using the circuit in Figure 5.1 [44], where the s-a-1 faults on lines x and y are intrinsically equivalent in M. There is another kind of equivalence, called extrinsic equivalence [44]. If two faults f_i and f_j located in a sub-circuit, M, are equivalent in the circuit N, but are not equivalent in the sub-circuit, then they are called extrinsically equivalent in M. In Figure 5.1, the s-a-1 faults on lines p and s are not equivalent in sub-circuit M. But these faults become equivalent in the circuit of Figure 5.1 and the relationship between the two faults is called extrinsic equivalence, which is a subset of functional equivalence. Such relationships are not detected by the hierarchical fault collapsing algorithm described in this chapter. A similar theorem can also be stated for diagnostic dominance as defined in Section 4.1.

In hierarchical fault collapsing, the collapsing information of sub-circuits is saved in a library. The information consists of a fault set, which includes equivalence collapsed set and faults on inputs and outputs of the sub-circuit, and their dominance

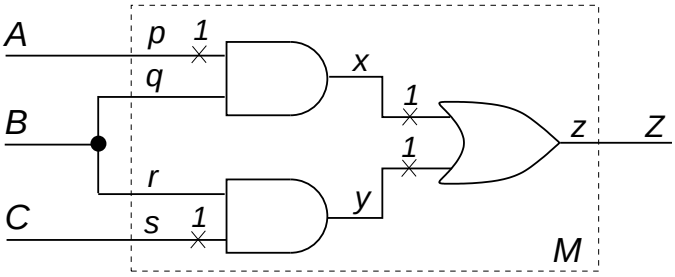

Figure 5.1: A circuit demonstrating extrinsic equivalence.

relations. The stuck-at faults on the input and output lines are added because these faults merge with the lines at higher levels of hierarchy as explained in Section 3.2.2 and help in further collapsing of faults. This also resolves the issue of embedding of sub-circuits as detailed in Section 3.2.2. So, in the collapsing procedure described later, all the instances of a sub-circuit are treated in the same way irrespective of the way it is embedded.

When collapsing faults hierarchically, there is a case that needs special attention. Let circuit N have an instance of circuit M. The line in N that is an input to the instance of M will be removed during expansion if this input in the circuit M is a fanout stem and the line in N is a fanout branch. Such an example is shown in Figure 5.2. The block named $G1$ in circuit N is an instance of the circuit M. On expansion, the fanout branch of B going into gate $G1$ is removed, so faults $G1(B, 0)$ and $G1(B, 1)$ are removed. Therefore, while saving the library information, if a fault on an input line is equivalent to another fault which is not on any input of the circuit, the latter fault is considered in equivalence collapsing. In Figure 5.2, the fault $b(1)$ in circuit M is functionally equivalent to fault $g1(1)$. In this case, the representative of this equivalent class should not be $b(1)$. If the fault on the input line is selected as the representative, $b(1)$ in this case, then on expansion, the fault $b(1)$ is removed leaving

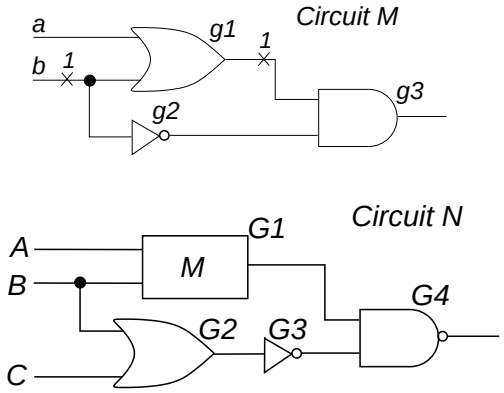

Figure 5.2: A hierarchical circuit.

no representative for the equivalence class consisting of fault $g1(1)$. This will lead to smaller, but erroneous, collapsed set. As explained earlier, the faults on the input and output lines are saved in the library file after equivalence collapsing is done. So, there will be two representatives of the same equivalence class in the library file, but one of them will be removed when the instance of the sub-circuit is considered for collapsing or when *algorithm equivalence* is applied at the top level.

5.1 Procedure to Hierarchically Collapse Faults

The collapsing starts at the topmost level of hierarchy. Only structural equivalence collapsing is done at this level. This is based on the line oriented structural fault collapsing technique as explained by Nadjarbashi *et al.* [67] with a few additions to suit the hierarchical collapsing. The criterion used in placing a specific stuck-at fault on a line is based on the gate driven by that line. Table 5.1 shows the faults to be placed, based on the gate the line is driving. Here fanout is treated as an actual component. Using this line collapsing technique, we obtain a structural equivalence

Table 5.1: Line fault collapsing.

Type of gate the line feeds into	Put this (these) fault(s) on the line
INV, BUF	none
OR, NOR	s-a-0
AND, NAND	s-a-1
Sub-circuit, Fanout	s-a-0, s-a-1
Primary Output	s-a-0, s-a-1

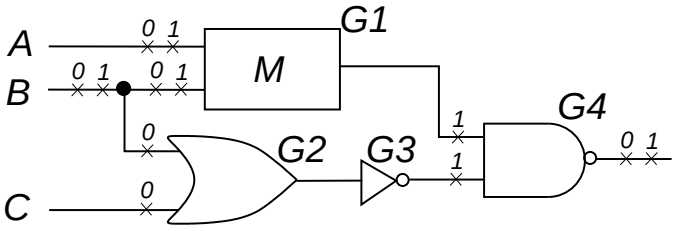

Figure 5.3: Circuit to demonstrate line collapsing technique.

collapsed set. This technique, when applied to the hierarchical circuit N of Figure 5.2, will result in 12 faults, which are shown in Figure 5.3.

A dominance matrix, as explained in Section 2.5, is created for the equivalence collapsed set. This matrix will now only have 1's on its diagonal and all the other entries of the matrix will be 0's. Next, the dominance relations between the faults in the equivalence collapsed set are found using the algorithm in the next section. For every s-a-0 on any input of an OR or NOR gate, the fault among the equivalent set that dominates it is found by setting b as 0 in the algorithm. For example, fault $C(0)$ in Figure 5.3 is dominated by the fault $G3(1)$. Then a 1 is introduced in the dominance matrix to indicate this dominance using the *update* algorithm [33, 34], which computes the transitive closure of the dominance matrix. A similar procedure is followed for s-a-1 faults on the inputs of AND and NAND gates.

5.1.1 Algorithm to Find the Dominance Matrix and its Transitive Closure

1. Consider a stuck-at-b fault (f_1) in the equivalence collapsed set at the input of a Boolean gate.

2. If the gate is of inverting type (NOT, NOR, NAND), then invert b.

3. If the equivalent set has s-a-b on this gate output, say f_2, then place a 1 at the intersection of the row corresponding to f_1 and column corresponding to f_2 and use *update* [33, 34] for computing transitive closure. Go to step 1 until all faults in the equivalence collapsed set that are on the inputs of Boolean gates are considered.

4. Move one gate forward towards the primary output and go to step 2.

For hierarchical fault collapsing, we need to have both stuck-at faults on inputs and outputs of all the instances of sub-circuits for reasons explained earlier in this chapter. By following Table 5.1, both stuck-at faults on the inputs of all instances are included. For outputs, the faults that are not present in the equivalence collapsed set are added. For these newly included faults, there will be an equivalent fault in the collapsed set. In Figure 5.3, fault $G1(0)$ is added and this fault is equivalent to fault $G4(1)$. This equivalent fault is also found using the algorithm in Section 5.1.1 by setting b as the stuck-at value of the added fault, 0 in this case, in step 2. There will be two diagonally symmetrical 1's introduced in step 3 of the algorithm to indicate this equivalence.

Next, the processing of instances of the sub-circuits is done. If the sub-circuit has a library file containing its collapsed information, the file is used to introduce

new faults and relations into the dominance matrix. If there is no such library file, a library file is created dynamically as follows. If the sub-circuit does not have any user-defined gates, that is instances of other sub-circuits, in its description and has less than a pre-specified number of gates, say 15, we can use the functional collapsing algorithm of Section 4.2.1. Otherwise, the sub-circuit is processed recursively in the same way as the top level description is processed. This is continued until all sub-circuits are processed. Then the collapsed fault set description is written to a file of the library for future use.

To obtain the equivalence collapsed set, the fault set is processed by *algorithm equivalence* which is quoted earlier in Section 3.2.1. This step removes all extra faults that were added to assist the hierarchical fault collapsing. If dominance collapsing is required, then *algorithm dominance*, as described in Section 3.2.1, is used. The basic idea of this procedure is same as that implemented by Prasad *et al.* [75].

The above procedure is applied to the hierarchical circuit in Figure 5.3. The sub-circuit M, being a smaller circuit, is subjected to functional collapsing. The collapsed information of sub-circuit M as saved in the library is detailed in Appendix A. The hierarchical collapsing results in sets of 11 faults for equivalence collapsing and 8 faults for dominance collapsing (includes one redundant fault $G1.g1(b,\ 0)$). These values can be compared with 12 and 8 when structural collapsing is used for the flattened circuit. The reduction is less because the circuit in Figure 5.3 has only one instance of a sub-circuit.

It should be noted that hierarchical collapsing results in much better collapse ratios if there are many instances of sub-circuits that have fewer than some pre-specified number of gates. The reduction is caused by the functional collapsing done in those smaller sub-circuits. Also, hierarchical collapsing takes less time, if there are multiple instances of the same sub-circuits or if the sub-circuit's collapsed information is already available in the library. If only structurally collapsed library information is used for all instances of sub-circuits, we get collapsing results that are exactly same as those of structural collapsing of the flattened circuit. But, it will take less CPU time than the structural collapsing.

While using the functional collapsing technique for smaller sub-circuits, the scheme of Figure 4.4(b), which corresponds to detection criterion, can only be used if the outputs of the instance of the sub-circuit are all primary outputs at the top level of the hierarchy. This is because, detection equivalence or dominance does not satisfy the theorem quoted at the beginning of this chapter, only the diagnostic relations do. If detection relations are used for at least one instance of any sub-circuit, then the collapsing should be called detection collapsing, else it can be called as diagnostic collapsing.

5.2 Results

In this section, we present the results of hierarchical fault collapsing, comparing the collapse ratios and collapsing time. The library consists of files containing the collapsed information of frequently used circuits like multiplexer, XOR, half adder, full adder, etc. Since the dominance matrix has very few 1's, it is implemented using

Table 5.2: Hierarchical fault collapsed sets.

Circuit name	All faults	No. of collapsed faults			
		Flattened (structural)		Hierarchical (functional)	
		Equiv.	Dom.	Equiv.	Dom.
Full adder	60	38 (0.63)	30 (0.50)	26 (0.43)	12 (0.20)
4-bit adder	234	146 (0.62)	114 (0,49)	98 (0.42)	48 (0.21)
16-bit adder	930	578 (0.62)	450 (0.48)	386 (0.42)	192 (0.21)
64-bit adder	3714	2306 (0.62)	1794 (0.48)	1538 (0.41)	768 (0.21)
256-bit adder	14850	9218 (0.62)	7170 (0.48)	6146 (0.41)	3072 (0.21)
1024-bit adder	59394	36866 (0.62)	28674 (0.48)	24578 (0.41)	12288 (0.21)
c432	1116	632 (0.56)	503 (0.45)	524 (0.47)	413 (0.37)
c499	2646	1574 (0.59)	1210 (0.46)	950 (0.36)	690 (0.26)

a sparse matrix representation [52]. The *algorithm equivalence* and *algorithm dominance* of Section 3.2.1 are slightly modified to suit the sparse matrix representation. The transitive closure is implemented using the *update* algorithm as described by Dave *et al.* [33, 34]. This algorithm takes time which grows linearly with the circuit size for sparse matrices, but in the worst case its complexity will be $O(n^3)$, where n is the number of gates in the circuit.

5.2.1 Comparison of Collapse Ratios

The collapsed fault set sizes obtained with hierarchical fault collapsing for different circuits are shown in Table 5.2. Those are compared with structural collapsing results for the corresponding flattened circuits obtained using Hitec [68] and Fastest [56] programs. The 4-bit adder consists of four full adders, the 16-bit adder has four 4-bit adders, and so on. The full adder circuit shown in the table is not a hierarchical circuit and the collapsing results of this circuit shown under the column Hierarchical are functional (diagnostic) collapsing results. The ISCAS'85 circuits [22] c432 and c499 have XOR gates which are considered as user-defined gates.

The collapse ratio of each collapsed set is shown by the accompanying fraction in parenthesis. There is considerable improvement in the collapse ratios when hierarchical collapsing is used. The reduction achieved for XOR and full adder sub-circuits using functional collapsing is passed onto other levels because of hierarchical collapsing. Even if all hierarchical adder circuits are described using only full adders, for example, a 1024-bit hierarchical adder is built using 1024 full adder sub-circuits, the collapse ratios would be the same. This is because, the reduction in these hierarchical adders is solely due to the functional collapsing done up to the full adder level. If the library files used for XOR and full adder circuits had structural, instead of functional, collapsed set information, the hierarchical fault collapsed sets would have the same collapse ratios as obtained for flattened circuits. Functional collapsing here refers to diagnostic collapsing. If detection collapsing were to be used, in any hierarchical adder, we could have used it on only one sub-circuit, the one which has all its outputs as primary outputs. This causes a reduction of 3 and 6 faults in any of the equivalence and dominance collapsed sets, respectively, under the column Hierarchical in Table 5.2. As explained in Chapter 4, these are the differences between diagnostic and detection collapsed sets for equivalence and dominance, respectively, of a full adder.

5.2.2 Comparison of CPU Times of Fault Collapsing

We have shown that hierarchical collapsing results in better (lower) collapse ratios. Now, we show that the CPU time taken by hierarchical collapsing is also lower than that required for flattened (structural) collapsing. A C program implementing

the procedure described in Section 5.1 is used to collapse the faults in both flattened and hierarchical circuits. The time reported in seconds in all the tables in this section is clocked on a 360MHz Sun UltraSparc 5_10 machine with 128MB memory. The description of the C program appears in Appendix B.

First, the implementation efficiency of our program is checked. We used differently sized flattened adder circuits (logic gate level circuits without any sub-circuits). The CPU time taken by our program for collapsing the faults is either similar to or better than that taken by other available programs [56, 68, 80]. In Table 5.3, we compare with the time taken by AUSIM [80] and Hitec [68]. Hitec and AUSIM do only equivalence collapsing while our program also provides dominance collapsed set. It should be noted that Hitec, in addition, calculates the controllabilities and observabilities for each line, which are later used in test generation. All the programs resulted in the same equivalence collapsed set sizes for all circuits. We do not have the CPU time taken by Hitec for the flattened circuit of 8192-bit adder because Hitec could not complete the collapsing operation for this circuit, probably because of some internal limit on size of the circuit. It can be seen from the table that our program takes considerably less CPU time than the other two programs.

The CPU times shown in Table 5.3 are plotted on the graph in Figure 5.4. Both axes of the plot have logarithmic scales with base 2. The portion of the plot corresponding to adders smaller than the 64-bit adder is included here only for completeness and can be neglected because of the coarse resolution problems associated

Table 5.3: Fault collapsing time for flattened circuits.

Circuit name	CPU time (seconds)		
	AUSIM	Hitec	Our Program
4-bit adder	0.08	0.23	0.02
8-bit adder	0.11	0.24	0.03
16-bit adder	0.20	0.30	0.04
32-bit adder	0.43	0.36	0.10
64-bit adder	1.10	0.57	0.24
128-bit adder	2.65	1.47	0.75
256-bit adder	9.60	5.09	2.49
512-bit adder	39.5	19.5	9.38
1024-bit adder	165	77.7	39.9
2048-bit adder	650	326	166.4
4096-bit adder	2623	1258	674.1
8192-bit adder	10681	–	2676

with the time commands used to clock the time. Since all the curves have a slope approximately equal to 2, it shows that the CPU time in all these cases is proportional to the square of the circuit size. This can be seen in Table 5.3.

To understand the complexity of our program, we measured the time taken for different functions involved in collapsing. The time taken by different sections of our program for flattened circuits is tabulated in Table 5.4. The column Flat Structure Processing gives the time required for building the structure of circuit. The columns Equivalence and Dominance Collapsing show the time taken for collapsing faults based on structural equivalence and dominance collapsing, respectively. The total time taken for collapsing is shown in the last column of Table 5.4 and is same as the last column of Table 5.3.

When the structure of the circuit is built, any node representing a gate or primary input has links to all nodes in its fanin and fanout lists. The time taken to build such a structure is proportional to the square of the circuit size. This is confirmed by

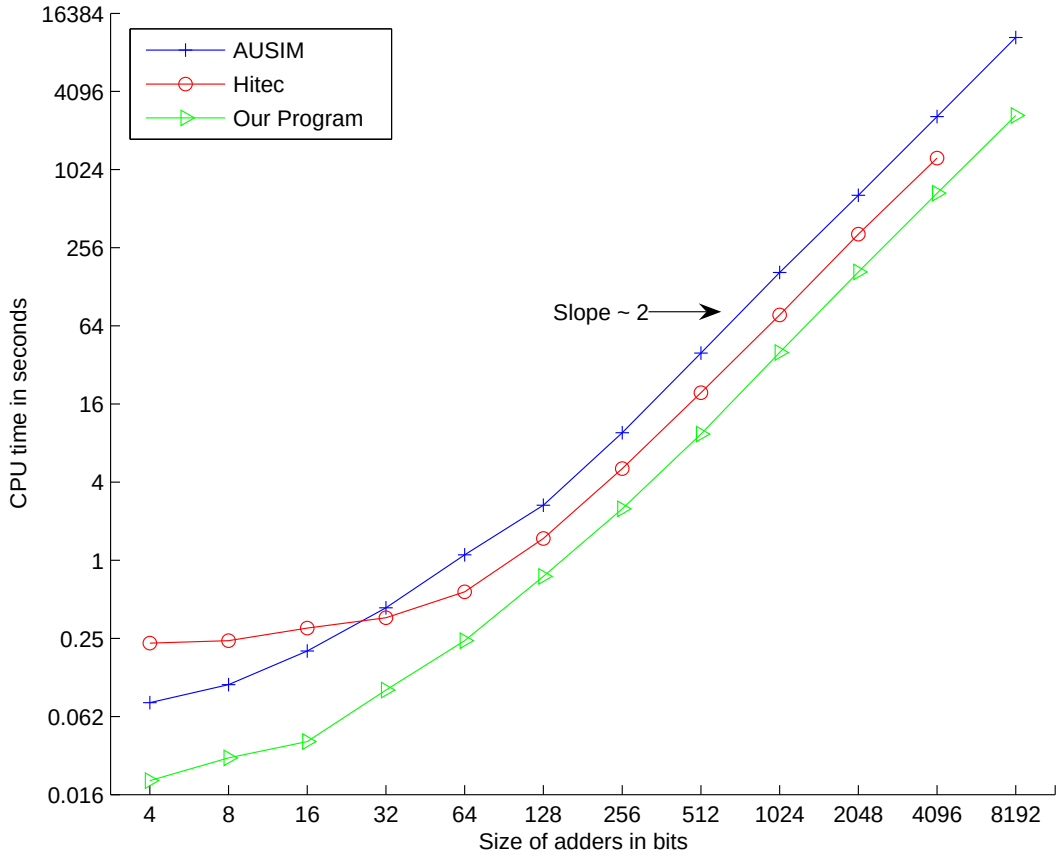

Figure 5.4: Fault collapsing time for flattened circuits.

Table 5.4: CPU time taken by different sections of our program for flattened circuits.

Circuit name	CPU time (seconds)			
	Flat Structure Processing	Equivalence Collapsing	Dominance Collapsing	Total
4-bit adder	0.0	0.0	0.01	0.02
8-bit adder	0.0	0.0	0.01	0.03
16-bit adder	0.02	0.0	0.01	0.04
32-bit adder	0.05	0.01	0.01	0.10
64-bit adder	0.13	0.02	0.03	0.24
128-bit adder	0.54	0.05	0.05	0.75
256-bit adder	2.05	0.09	0.09	2.49
512-bit adder	8.60	0.17	0.20	9.38
1024-bit adder	38.3	0.36	0.41	39.9
2048-bit adder	163.1	0.74	0.84	166.4
4096-bit adder	667.4	1.49	1.63	674.1
8192-bit adder	2662	3.43	3.72	2676

the time taken by our program to build the structure of the flattened circuits, as shown in the column Flat Structure Processing in Table 5.4. The time taken for equivalence and dominance collapsing grows linearly with the circuit size. This is also verified using Figure 5.5, which is a plot of the data in Table 5.4 for 64-bit and larger adders. The slope of the curves corresponding to Equivalence Collapsing and Dominance Collapsing is 1, while that of Flat Structure Processing and Total is 2. The plot also shows that, for large circuits, the total CPU time is dominated by the time needed to build the circuit structure. Hence, for large circuits, the total time required for collapsing grows as the square of the circuit size. Similar conclusions are drawn for the time taken for collapsing using Hitec.

The times taken by different commands used for collapsing in Hitec are shown in Table 5.5. The structure of the circuit is built using the internal routine *level* and the time taken by this command for different circuits is shown under the column

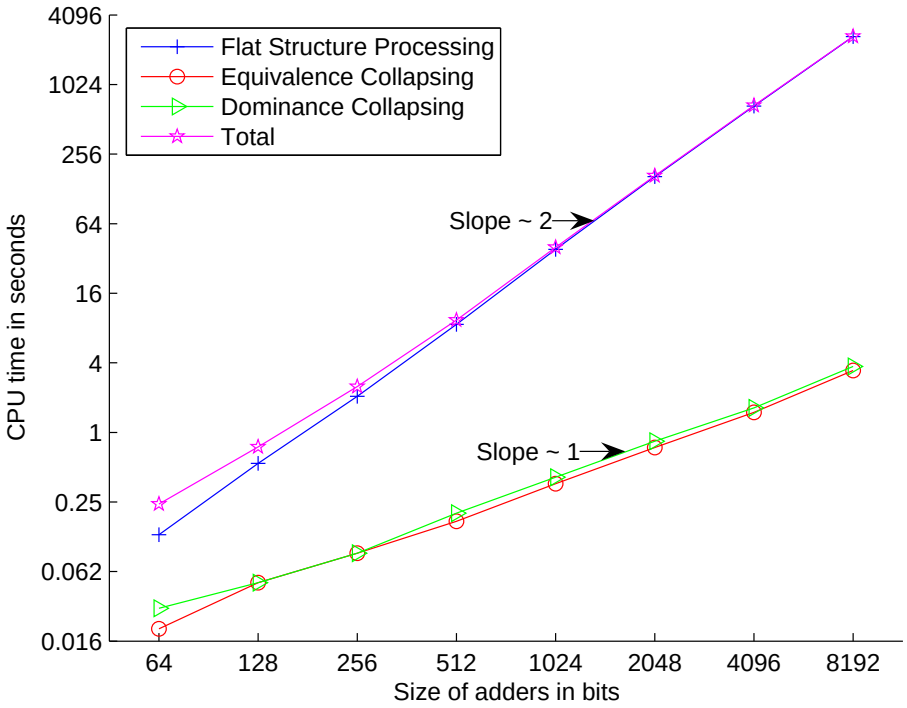

Figure 5.5: CPU time taken by different sections of our program for flattened circuits.

Table 5.5: CPU time taken by different commands of Hitec for collapsing faults.

Circuit name	CPU time (seconds)		
	Structure Processing (*level*)	Equivalence Collapsing (*equiv*)	Total
4-bit adder	0.10	0.06	0.23
8-bit adder	0.12	0.06	0.24
16-bit adder	0.14	0.09	0.30
32-bit adder	0.17	0.13	0.36
64-bit adder	0.32	0.16	0.57
128-bit adder	1.03	0.34	1.47
256-bit adder	4.0	0.88	5.09
512-bit adder	16.0	3.15	19.52
1024-bit adder	64.9	12.2	77.7
2048-bit adder	275.1	50.4	326
4096-bit adder	1045.4	210	1258

Structure Processing (*level*). The column Equivalence Collapsing gives the time taken for equivalence collapsing using the routine *equiv*. Total gives the total time taken for collapsing. A comparison of the CPU times in Tables 5.4 and 5.5 is shown in Figure 5.6. The times taken by different commands of Hitec are shown in solid line curves, while the times taken by our program are shown in dashed line curves. We note the difference in the slopes of the plots corresponding to equivalence collapsing. Our program does it in time proportional to the circuit size (slope ≈ 1), while Hitec time grows at a rate proportional to the square of the circuit size (slope ≈ 2). The time taken by our program for equivalence collapsing grows linearly because we use the line oriented collapsing technique explained in Section 5.1, which is of linear complexity in the size of the circuit. Our program just needs one pass from primary inputs to outputs to determine the equivalence collapsed set. We are not able to explain why Hitec takes time which is proportional to the square of the circuit size

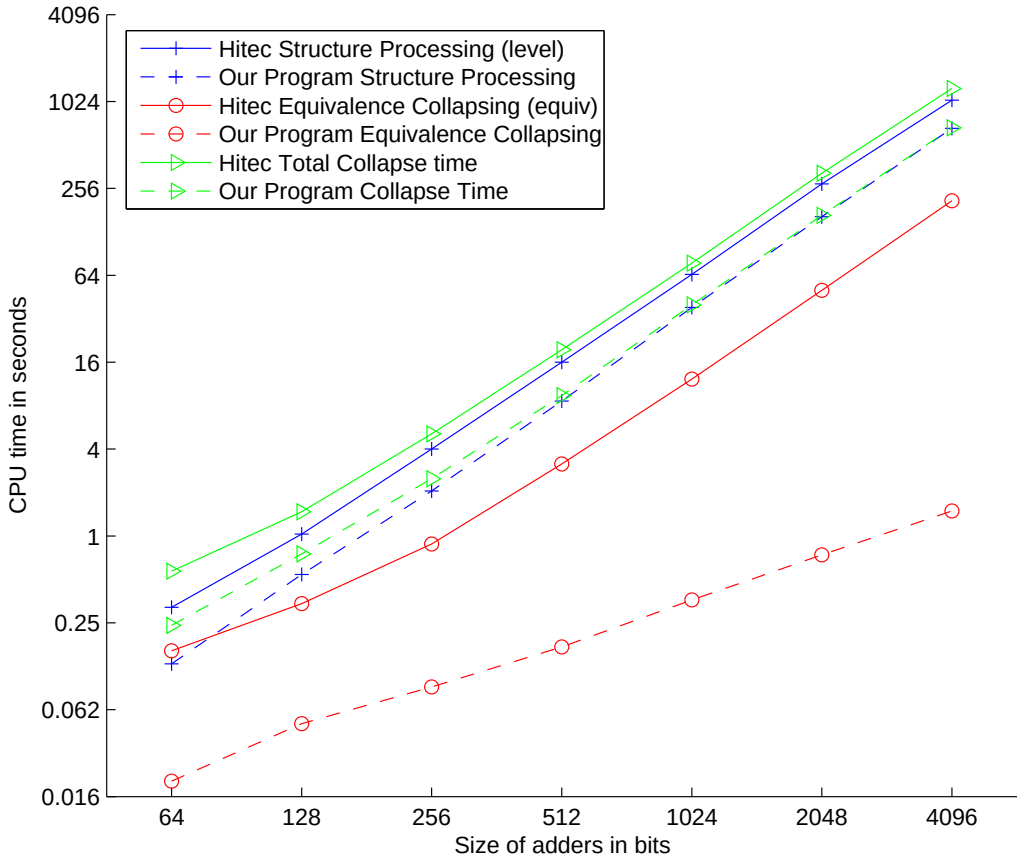

Figure 5.6: Comparison of CPU times taken by Hitec and our program.

for equivalence collapsing. The times taken by AUSIM [80] and Gentest [30] for equivalence collapsing are also linearly proportional to the circuit size.

Next, our program is used to collapse faults in the circuits described hierarchically with many levels just like the hierarchical adder circuits in Table 5.2, i.e., a 1024-bit adder is built using four 256-bit adders, which are built using four 64-bit adders, and so on. The times clocked by different functions involved in the program are shown in Table 5.6. The column Hierarchical Structure Processing gives the time taken to build the structure of the hierarchical circuit. The time required for both equivalence

Table 5.6: CPU time taken by different sections of our program for hierarchical circuits.

Circuit name	CPU time (seconds)			
	Hierarchical Structure Processing	Equiv.+Dom. Collapsing	Library	Total
4-bit adder	0.0	0.0	0.0	0.01
8-bit adder	0.0	0.0	0.01	0.02
16-bit adder	0.01	0.01	0.02	0.05
32-bit adder	0.01	0.01	0.03	0.07
64-bit adder	0.01	0.01	0.07	0.10
128-bit adder	0.03	0.02	0.13	0.24
256-bit adder	0.05	0.02	0.19	0.49
512-bit adder	0.17	0.04	0.36	1.05
1024-bit adder	0.55	0.08	0.73	2.31
2048-bit adder	2.10	0.20	1.52	4.80
4096-bit adder	9.25	0.37	3.1	16.6
8192-bit adder	40.1	0.79	6.0	55.0

and dominance collapsing is shown under the column Equiv.+Dom. Collapsing, of which the time required for equivalence collapsing is the major component. The time taken to introduce new faults of the sub-circuits from library files and their relations is shown under the column Library. This time was negligible in Table 5.4, since there were no sub-circuits involved. The column Total gives the total time taken by the program which includes the time taken to dynamically collapse all the sub-circuits used in the hierarchical circuit. However, the one-bit full adder block which has been earlier collapsed using the functional technique is assumed to be available from a stored library. For example, the time shown against the 1024-bit adder includes the time of building the library files for 1024-bit, 256-bit, 64-bit, 16-bit and 4-bit adders, but not that of the 1-bit adder library element. It can be seen that, even here the time required to build the circuit dominates as the size of the circuit grows.

We compare the times taken by our program for hierarchical circuits and flattened circuits in Figure 5.7. Here we compare the times taken for building the structure and collapsing the faults, though the collapse ratios obtained for hierarchical circuits are better (lower) than those for flattened circuits. The times taken by our program for flattened circuits are shown in solid line curves, while the times taken for hierarchical circuits are shown in dashed line curves. The collapsing time for flattened circuits is the sum of Equivalence and Dominance Collapsing columns of Table 5.4. The collapsing time for hierarchical circuits is the sum of Equiv.+Dom. Collapsing and Library columns of Table 5.6. It can be seen from Figure 5.7 that the time taken for collapsing in both cases grows linearly with the circuit size. But there is considerable improvement in the CPU time used to build the circuit. When we build the hierarchical structure, the internal nodes of sub-circuits are considered only once and at a lower level of hierarchy. But, in flattened circuits, the internal nodes of all the sub-circuits have to be considered at the same level. So, the number of nodes in the hierarchical structure is much lower than that in the flat structure. This is the reason why the hierarchical structure is built much faster than the flat structure. This improvement shows up in the comparison of total times (curves drawn through triangles) for the two cases in Figure 5.7.

In Figure 5.7, the time taken to build the structure of a hierarchical circuit grows at a rate proportional to the square of the circuit size. It is expected that this complexity could be less than the square. This is because, in the circuits that we considered, different sized ripple carry adders, the number of inputs and outputs grow at the same rate as the circuit size. But, according to Rent's rule [24, 82], the

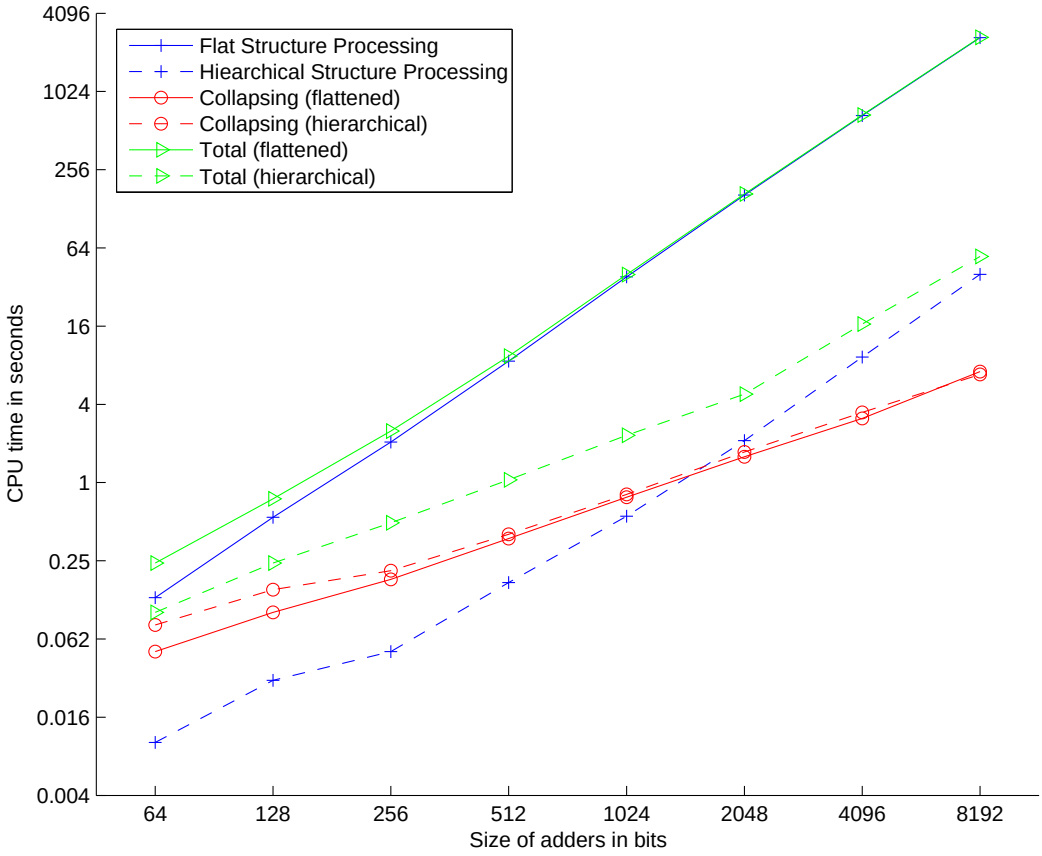

Figure 5.7: Comparison of CPU time taken by our program for hierarchical and flattened circuits.

number of input and output terminals for a typical block containing G logic gates is given by Equation 5.1:

$$T = K \times G^{\alpha} \tag{5.1}$$

where K is a constant between 1 and 5, and the exponent α lies in the range 0.5 to 0.67. For the ripple carry adders, the exponent α is close to 1.

The time taken to build the structure of the circuit is proportional to square of the number of gates and primary inputs in the circuit, as shown by the curves drawn through +'s in Figure 5.6. For a hierarchical circuit, the number of nodes that represent the structure of the circuit is dominated by the number of inputs and outputs, as the internal nodes of all sub-circuits are considered at a lower level of hierarchy. For a general hierarchical circuit, which obeys Rent's rule, the time taken to build the circuit should grow at a rate twice of α. If we assume $\alpha = 0.5$, then the time taken to build the circuit grows linearly with the circuit size. So, for a typical hierarchical circuit, the total time required for collapsing should be near to linear complexity in the circuit size.

We now compare the CPU time required for collapsing circuits with different levels of hierarchy. The Table 5.7 shows three hierarchical versions of the adders. The values in the last column correspond to the circuits described hierarchically with many levels, just like the hierarchical circuits in Table 5.6, i.e., a 1024-bit adder is built using four 256-bit adders, and so on. The two-level circuits are the circuits with a hierarchical depth of 2. These circuits are built using a full adder as the sub-circuit, i.e., a 1024-bit adder is built using 1024 full adders. The flattened circuits have a hierarchical depth of 1, i.e., they are described completely at the gate level.

Table 5.7: CPU time improvement by hierarchy.

Circuit name	Flattened Circuit		Hierarchical Circuit	
	Hitec	Our Program	Two-level	Multi-level
4-bit adder	0.23	0.02	0.01	0.01
8-bit adder	0.24	0.03	0.02	0.02
16-bit adder	0.30	0.04	0.05	0.05
32-bit adder	0.36	0.10	0.08	0.07
64-bit adder	0.57	0.24	0.16	0.10
128-bit adder	1.47	0.75	0.32	0.24
256-bit adder	5.09	2.49	0.69	0.49
512-bit adder	19.5	9.38	1.52	1.05
1024-bit adder	77.7	39.9	3.60	2.31
2048-bit adder	326	166.4	10.3	4.80
4096-bit adder	1258	674.1	35.1	16.6
8192-bit adder	–	2676	127.2	55.0

It has been pointed out in Section 5.2.1 that the collapse ratios obtained for the hierarchical circuits (i.e., two-level and multi-level) in Table 5.7 are the same. The data shown in Table 5.7 corresponding to 64-bit and larger adders is plotted in Figure 5.8. The time values in the last (multi-level) column include the time required to build the library file of collapsed sub-circuits at all levels. It can be observed from the figure that the circuits described using greater depth of hierarchy require less time for collapsing. When a different hierarchy was used, say a 1024-bit adder was built using eight 128-bit adders and the 128-bit adder used eight 16-bit adders, and so on, it resulted in similar times as reported in the last column. However, for another hierarchy when the 1024-bit adder was built using sixteen 64-bit adders, the 64-bit adder used sixteen 4-bit adders, and the 4-bit adder used four full adders, it required less time than the two-level circuit, but more than the multi-level circuit of Table 5.7.

If we assume that the library has the collapsing information for all sub-circuits used in the circuit, then the time required for collapsing monotonically decreases

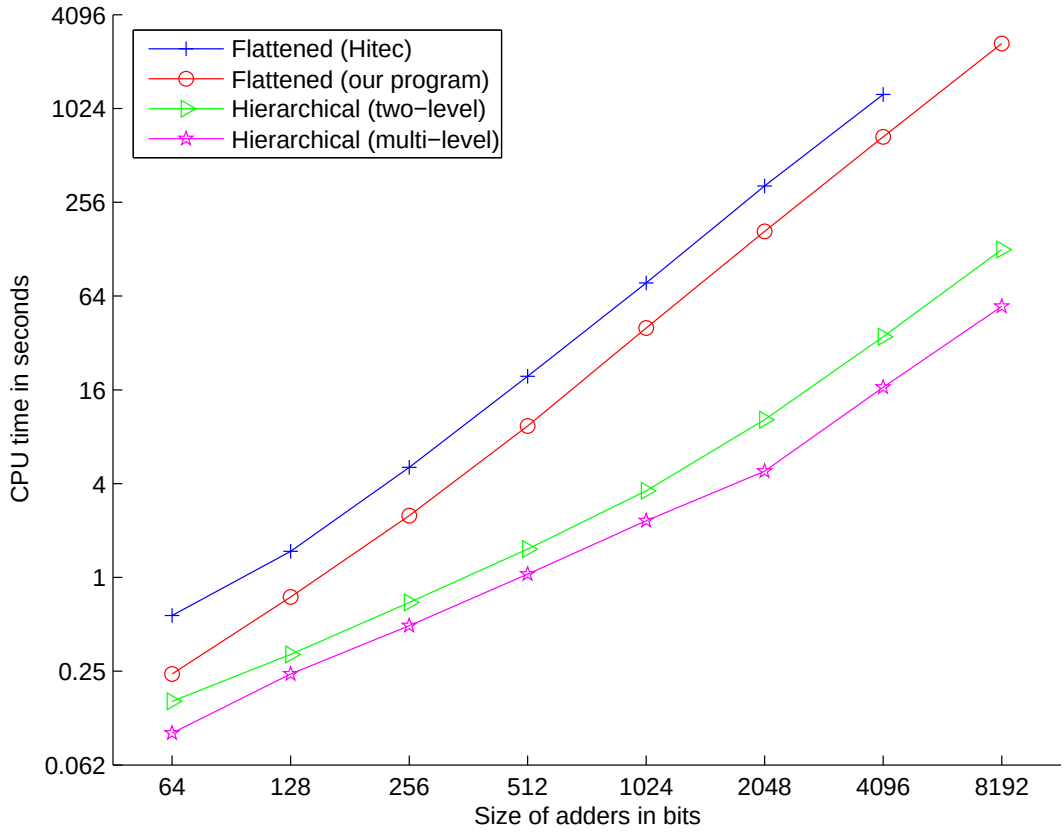

Figure 5.8: CPU time improvement by hierarchy.

as the number of levels of hierarchy increases. But, when we have to build the collapsing information of the sub-circuits dynamically, we should take care that the time savings achieved by increasing the number of levels is not offset by the time required to collapse the sub-circuits. For example, when the library contains the collapsing information on 4096-bit and 2048-bit adders, a 8192-bit adder built using four 2048-bit adders is collapsed in 50.2 seconds, while a 8192-bit adder using two 4096-bit adders takes 50.0 seconds. The savings of 0.2 seconds is easily offset by the difference (11.8 seconds) between the times taken to build the library files for the sub-circuits, 4096-bit adder (16.6 seconds) and 2048-bit adder (4.8 seconds). So, when we have to dynamically build the library information of the sub-circuits, the time may monotonically decrease up to certain number of levels of hierarchy, depending upon the specific circuits.

As we increase the number of levels in the hierarchy, the number of nodes in the structure is decreased. But, the percentage reduction in the number of nodes decreases with increasing number of levels. So, the percentage saving in time decreases as the number of levels increase. This can be seen from the data in Table 5.8. In this table, we consider the collapse times of three circuits, 2048-bit, 4096-bit and 8192-bit adders. Each adder is built using only one type of sub-circuit, namely 1-bit, 4-bit, 16-bit, 64-bit, 256-bit or 1024-bit adder. The time required for collapsing faults in each case is shown in Table 5.8. The columns in the table correspond to the sub-circuits used to build the circuit. For example, the 2048-bit adder takes 5.93 seconds when built using only 4-bit adders. It is assumed that the collapsing information of 1-bit, 4-bit, 16-bit, 64-bit, 256-bit and 1024-bit adders is present in the library. It is clearly

Table 5.8: CPU time taken by our program for different levels of hierarchy.

Circuit name	CPU time (seconds)					
	1-bit	4-bit	16-bit	64-bit	256-bit	1024-bit
2048-bit adder	10.3	5.93	4.90	4.80	4.75	4.72
4096-bit adder	35.1	18.7	15.4	14.6	14.4	14.3
8192-bit adder	127.2	66.4	57.1	53.6	51.4	50.5

seen that the reduction in time levels off as we move from left to right through the table.

5.3 Summary

In this chapter, we have explained the hierarchical fault collapsing technique and the results obtained with this technique. We have shown that the advantage of smaller collapse ratios achieved using functional collapsing techniques for smaller sub-circuits is passed on to the larger hierarchical circuits. The time required for collapsing a hierarchical circuit is less than that for the corresponding flat circuit. The time taken for flat circuits is proportional to the square of the circuit size, while for hierarchical circuits, it is shown that the time for collapsing could be reduced to lower complexities, near to linear complexity. As the number of levels of hierarchy in the circuit increases, the time required for collapsing decreases.

Chapter 6

Conclusions

Several functional collapsing techniques are presented in Chapters 3 and 4. Though functional collapsing produces low collapse ratios, it takes more CPU time, as functional collapsing is an NP-hard task. Hierarchical collapsing, which is explained in Chapter 5, is faster but does not allow functional collapsing through all levels of hierarchy. So, hierarchical collapsing combined with functional collapsing at lower levels gives the best result in terms of both CPU time and collapse ratio.

The conventional structural collapsing techniques generally yield a collapse ratio of about 50%. When functional (detection) dominance collapsing is used, the collapse ratio is reduced to the range of 10 to 20%. The fault set sizes obtained using the algorithms in Chapter 4 are considerably smaller than the previously reported results. The advantage of such a small collapse ratio may be in the reduction of fault simulation effort and in the number of test vectors, though the latter conclusion is not clearly reflected in the results of Table 4.3. Also, fault diagnosis is easier with smaller fault sets. There are methods [16, 72] that aim at obtaining, though with no guarantee of, the smallest vector set to detect a given fault set. The number of test vectors when compared to the lower bound given by a method due to Akers and Krishnamurthy [12] indicates that further reduction is possible. Berger and Kohavi [17] establish the lower bound on the necessary number of fault-detecting tests in fanout free combinational circuits, which is rarely applicable to real circuits.

Hierarchical fault collapsing, which is described in Chapter 5, is shown to take less time than structural fault collapsing for the corresponding flat circuits. When functional techniques are used for collapsing the sub-circuits, then hierarchical fault collapsing also results in lower collapse ratios than those of structural collapsing. A significant result of this work is that the time for hierarchical fault collapsing decreases as the depth of hierarchy increases.

Care is needed in using dominance collapsed set since there can be instances where the dominated fault is redundant and the dominating fault (not included in the collapsed set) is testable. One such case is presented in [2] to point out that checkpoint faults, which are based on dominance, are not sufficient target faults for test generation. For fault diagnosis, we can only use the diagnostic equivalence collapsed set.

6.1 Future Work

Any Boolean circuit can be partitioned based on the library cells using Mentor Graphics tools. The fault collapsing library of these standard cells can be generated and hierarchical fault collapsing can be used for the partitioned circuit. We can incorporate VHDL or Verilog input for hierarchical netlist.

We have used an ATPG to find redundancies. There are recent methods [6, 34, 39, 54, 65] of redundancy identification, not based on ATPG but using non-exhaustive search, that are less time consuming. Their use may provide trade offs between the CPU time and efficiency for the detection of redundancies used in our

functional dominance algorithm. However, these techniques may not result in the smallest possible collapsed set.

There are existing programs that attempt to find small test sets [72]. However, no known program has produced the 12-vector test set for the 4-bit ALU circuit [48]. So, there is scope for future work.

A customized ATPG can be written to find a minimal test vector set required to detect the collapsed fault set obtained using the algorithms and thereby detect all the faults in a given circuit. A possible strategy might be to run ATPG for the faults ordered with the hardest to detect fault placed first. Selecting the right vector from the set of all vectors that detect the target fault may be the key to finding the minimal set of vectors. While this problem has not been solved, we may suggest a probable heuristic. After a fault is detected, among all the test vectors that detect that fault, the one which has higher probability of detecting other faults may be selected. Then a fault simulator would be run with this test vector for fault dropping. This would probably lead to the least number of test vectors required to detect all the faults in the circuit.

The fault collapsing techniques presented here are only for combinational circuits. They can be extended to sequential circuits, using the techniques described in Section 2.3. The application of the presented fault collapsing techniques is only for stuck-at faults. The development of similar techniques for other fault models can be further investigated.

Bibliography

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Piscataway, New Jersey: IEEE Press, 1990.

[2] M. Abramovici, P. R. Menon, and D. T. Miller, "Checkpoint Faults are not Sufficient Target Faults for Test Generation," *IEEE Trans. Computers*, vol. C-35, no. 8, pp. 769–771, Aug. 1986.

[3] M. Abramovici, D. T. Miller, and R. K. Roy, "Dynamic Redundancy Identification in Automatic Test Generation," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 3, pp. 404–407, Mar. 1992.

[4] V. D. Agrawal, "Sampling Techniques for Determining Fault Coverage in LSI Circuits," *Journal of Digital Systems*, vol. 5, no. 3, pp. 189–202, 1981.

[5] V. D. Agrawal and P. Agrawal, "An Automatic Test Generation System for Illiac IV Logic Boards," *IEEE Trans. Computers*, vol. C-21, no. 9, pp. 1015–1017, Sept. 1972.

[6] V. D. Agrawal, M. L. Bushnell, and Q. Lin, "Redundancy Identification Using Transitive Closure," *Proc. IEEE* 5^{th} *Asian Test Symposium*, Nov. 1996, pp. 4–9.

[7] V. D. Agrawal and H. Kato, "Fault Sampling Revisited," *IEEE Design & Test of Computers*, vol. 7, pp. 32–35, Aug. 1990.

[8] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre, "Fault Collapsing via Functional Dominance," *Proc. International Test Conf.*, 2003, pp. 274–280.

[9] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre, "It is Sufficient to Test 25-Percent of Faults," *Proc. IEEE* 7^{th} *VLSI Design & Test Workshops*, Aug. 2003, pp. 368–374.

[10] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Reading, Massachusetts: Addison-Wesley, 1987.

[11] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," *Proc. International Test Conf.*, 1987, pp. 1100–1107.

[12] S. B. Akers and B. Krishnamurthy, "Test Counting: A Tool for VLSI Testing," *IEEE Design & Test of Computers*, vol. 6, no. 5, pp. 58–77, Oct. 1989.

[13] S. A. Al-Arian and M. A. Al-Kharji, "Fault Simulation and Test Generation by Fault Sampling Techniques," *Proc. International Conf. on Computer Design*, Oct. 1992, pp. 365–368.

[14] H. Al-Asaad and R. Lee, "Simulation-Based Approximate Global Fault Collapsing," *Proc. International Conf. on VLSI*, 2002, pp. 72–77.

[15] M. E. Amyeen, W. K. Fuchs, I. Pomeranz, and V. Boppana, "Fault Equivalence Identification using Redundancy Information and Static and Dynamic Extraction," *Proc.* 19^{th} *IEEE VLSI Test Symp.*, 2001, pp. 124–130.

[16] C. Benmehrez and J. F. McDonald, "Measured Performance of a Programmed Implementation of the Subscripted D-Algorithm," *Proc.* 20^{th} *Design Automation Conf.*, June 1983, pp. 308–315.

[17] I. Berger and Z. Kohavi, "Fault Detection in Fanout Free combinational Networks," *IEEE Trans. Computers*, vol. C-22, no. 10, pp. 908–914, Oct. 1973.

[18] V. Boppana and W. K. Fuchs, "Dynamic Fault Collapsing and Diagnostic Test Pattern Generation for Sequential Circuits," *IEEE/ACM International Conference on CAD (ICCAD)*, Nov. 1998, pp. 147–154.

[19] D. C. Bossen and S. J. Hong, "Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks," *IEEE Trans. Computers*, vol. 11, no. C-20, pp. 1252–1257, Nov. 1971.

[20] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital Systems.* Woodland Hills, CA: Computer Science Press, 1976.

[21] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. IEEE International Symposium on Circuits and Systems*, May 1989, pp. 1929–1934.

[22] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuit and a target translation in FORTRAN," distributed on a tape to participants of the Special Session on ATPG and Fault Simulation, International Symposium on Circuits and Systems, June 1985; partially characterized in F. Brglez, P. Pownall, R. Hum, "Accelerated ATPG and Fault Grading via Testability Analysis," *Proc. IEEE International Symposium on Circuits and Systems*, June 1985, pp. 695-698, reprint of the article is available with the tape from MCNC. Website: http://www.cbl.ncsu.edu/CBL_Docs/Bench.html.

[23] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[24] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits.* Boston, MA: Kluwer Academic Publishers, 2000.

[25] S. T. Chakradhar, V. D. Agrawal, and M. L. Bushnell, "Polynomial Time Solvable Fault Detection Problems," *Proc.* 20^{th} *Fault-Tolerant Computing Symposium (FTCS-20)*, (Newcastle-upon-Tyne, UK), June 1990, pp. 56–63.

[26] S.-J. Chang and M. A. Breuer, "A Fault-collapsing Analysis in Sequential Logic Networks," *Bell Systems Technical Journal*, vol. 60, no. 9, pp. 2259–2271, Nov. 1981.

[27] J. E. Chen, C. L. Lee, and W. Z. Shen, "Circuit Example to Demonstrate that Fanout Stems of Primary Inputs must be Checkpoints," *IEEE Trans. Computers*, vol. 25, no. 25, pp. 1726–1728, Dec. 1989.

[28] J. E. Chen, C. L. Lee, and W. Z. Shen, "Checkpoints in Irredundant Two-Level Combinational Circuits," *Journal of Electronic Testing: Theory and Application*, vol. 2, no. 4, pp. 395–397, Nov. 1991.

[29] J. E. Chen, C. L. Lee, and W. Z. Shen, "Single-Fault Fault-Collapsing Analysis in Sequential Logic Circuits," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 12, pp. 1559–1568, Dec. 1991.

[30] W. T. Cheng and T. J. Chakraborty, "Gentest: An Automatic Test Generation System for Sequential Circuits," *Computer*, vol. 22, no. 4, pp. 43–49, April 1989.

[31] F. W. Clegg, "Use of SPOOF's in the Analysis of Faulty Logic Networks," *IEEE Trans. Computers*, vol. 22, no. 3, pp. 229–234, Mar. 1973.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, Massachusetts: MIT Press, 2 edition, 2001.

[33] K. K. Dave, "Using Contrapositive Rule to Enhance the Implication Graphs of Logic Circuits," Master's thesis, Rutgers University, New Brunswick, May 2004.

[34] K. K. Dave, V. D. Agrawal, and M. L. Bushnell, "Using Contrapositive Law in an Implication Graph to Identify Logic Redundancies," *Proc.* 18^{th} *International Conf. VLSI Design*, Jan. 2005, pp. 723–729.

[35] R. D. Eldred, "Test Routines Based on Symbolic Logical Statements," *Journal of the ACM*, vol. 6, no. 1, pp. 33–36, Jan. 1959.

[36] H. Fujiwara, "Computational Complexity of Controllability/Observability Problems for Combinational Circuits," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 762–767, June 1990.

[37] H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problem for Combinational Logic Circuits," *IEEE Trans. Computers*, vol. C-31, no. 6, pp. 555–560, June 1982.

[38] J. M. Galey, R. E. Noby, and J. P. Roth, "Techniques for the Diagnosis of Switching Circuit Failures," in R. S. Ledley, editor, *Proc. of the Second Annual Symposium on Switching Circuit Theory and Logical Design*, (Detroit), AIEE, Oct. 1961, pp. 152–160.

[39] V. Gaur, V. D. Agrawal, and M. L. Bushnell, "A New Transitive Closure Algorithm with Application to Redundancy Identification," *Proc.* 1^{st} *Int. Workshop on Electronic Design, Test and Applications (DELTA'02)*, (Christ Church, New Zealand), Jan. 2002, pp. 496–500.

[40] P. Goel, *The Feedforward Logic Model in the Testing of Large Scale Integrated Logic Circuits*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, Sept. 1973.

[41] P. Goel, "Test Generation Costs Analysis and Projections," *Proc.* 17^{th} *IEEE/ACM Design Automation Conf.*, June 1980, pp. 77–84.

[42] K. Gopalakrishnan and B. Bhattacharya, "Noncheckpoint Target Faults How to Order Them?," *Proc.* 34^{th} *Midwest Circuits and Systems Symposium*, May 1991, pp. 619–622.

[43] A. Goundan, *Fault Equivalence in Logic Networks*. PhD thesis, University of Southern California, Los Angeles, Feb. 1978.

[44] A. Goundan and J. P. Hayes, "Identification of Equivalent Faults in Logic Networks," *IEEE Trans. Computers*, vol. C-29, no. 11, pp. 978–985, Nov. 1980.

[45] T. Grüning, U. Mahlsdedt, and H. Koopmeiners, "DIATEST: A Fast Diagnostic Test Pattern Generator for Combinational Circuits," *Proc. International Conf. Computer-Aided Design*, Nov. 1991, pp. 194–197.

[46] R. Hahn, R. Krieger, and B. Becker, "A Hierarchical Approach to Fault Collapsing," *Proc. European Design & Test Conf.*, 1994, pp. 171–176.

[47] I. Hartanto, V. Boppana, and W. K. Fuchs, "Diagnostic Fault Equivalence Identification Using Redundancy Information & Structural Analysis," *Proc. International Test Conf.*, Oct. 1996, pp. 294–302.

[48] J. P. Hayes. website: http://www.eecs.umich.edu/~jhayes/iscas/74181.html.

[49] J. P. Hayes, "A Study of Digital Network Structure and its Relation to Fault Diagnosis," Technical Report R-467, CFSTI AD 707 691, Coordinated Science Laboratory, Univ. Illinois, Urbana-Champaign, May 1970.

[50] J. P. Hayes, "Modeling Faults in Digital Logic Circuits," in R. Saeks and S. R. Liberty, editors, *Rational Fault Analysis*, (NY), Marcel Dekker, 1977, pp. 78–95.

[51] J. P. Hayes and A. D. Friedman, "Test Point Placement to Simplify Fault Detection," *IEEE Trans. Computers*, vol. C-23, no. 7, pp. 727–735, July. 1974.

[52] E. Horowitz and S. Sahni, *Fundamentals of Data Structures in PASCAL*. New York, USA: Computer Science Press, Inc, 1984.

[53] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. Computers*, vol. C-24, pp. 242–247, Mar. 1975.

[54] M. A. Iyer and M. Abramovici, "FIRE: A Fault-Independent Combinational Redundancy Identification Algorithm," *IEEE Trans. VLSI Systems*, vol. 4, no. 2, pp. 295–301, June 1996.

[55] N. K. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge University Press, 2003.

[56] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," *IEEE Trans. Computers*, vol. 42, no. 11, pp. 1361–1371, Nov. 1993.

[57] B. Krishnamurthy and S. B. Akers, "On the Complexity of Estimating the Size of a Test Set," *IEEE Trans. Computers*, vol. C-33, no. 8, pp. 750–753, Aug. 1984.

[58] S. Kundu, "Pitfalls of Hierarchical Fault Simulation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 2, pp. 312–314, Feb. 2004.

[59] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Technical Report 12_93, Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[60] J. Lee and J. H. Patel, "Hierarchical Test Generation under Architectural Level Functional Constraints," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1144–1151, Sept. 1996.

[61] A. Lioy, "Looking for Functional Fault Equivalence," *Proc. International Test Conf.*, Oct. 1991, pp. 858–863.

[62] A. Lioy, "Advanced Fault Collapsing," *IEEE Design & Test of Computers*, vol. 9, no. 1, pp. 64–71, Mar. 1992.

[63] A. Lioy, "On the Equivalence of Fanout-Point Faults," *IEEE Trans. Computers*, vol. 42, no. 3, pp. 268–271, Mar. 1993.

[64] E. J. McCluskey and F. W. Clegg, "Fault Equivalence in Combinational Logic Networks," *IEEE Trans. Computers*, vol. C-20, no. 11, pp. 1286–1293, Nov. 1971.

[65] V. J. Mehta, K. K. Dave, V. D. Agrawal, and M. L. Bushnell, "A Fault-Independent Transitive Closure Algorithm for Redundancy Identification," *Proc.* 16^{th} *International Conf. VLSI Design*, Jan. 2003, pp. 149–154.

[66] A. Motohara, M. Murakami, M. Urano, Y. Masuda, and M. Sugano, "An approach to fast hierarchical fault simulation," *Proc.* 25^{th} *Design Automation Conference*, 1988, pp. 698–703.

[67] M. Nadjarbashi, Z. Navabi, and M. R. Movahedin, "Line Oriented Structural Equivalence Fault Collapsing," *IEEE Workshop on Model and Test*, 2000.

[68] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Design Automation Conference*, Feb. 1991, pp. 214–218.

[69] J. F. Poage, "Derivation of Optimum Tests to Detect Faults in Combinational Circuits," *Proc. Symposium on Mathematical Theory of Automata*, April 1962, pp. 483–528.

[70] J. F. Poage and E. J. McCluskey, "Derivation of Optimum Test Sequences for Sequential Machines," *Proc.* 5^{th} *Symposium on Switching Theory and Logic Design*, 1964, pp. 121–132.

[71] I. Pomeranz and Z. Kohavi, "The Minimum Test Set Problem for Circuits with Non-reconvergent Fanout," *Journal of Electronic Testing: Theory and Application*, vol. 2, no. 4, pp. 339–349, Nov. 1991.

[72] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, no. 7, pp. 1040–1049, July 1993.

[73] I. Pomeranz and S. M. Reddy, "Level of Similarity: A Metric for Fault Collapsing," *Proc. Design, Automation and Test in Europe Conf.*, volume 1, Feb. 2004, pp. 56–61.

[74] I. Pomeranz and S. M. Reddy, "On Fault Equivalence, Fault Dominance and Incompletely Specified Test Sets," *IEEE Trans. Computer-Aided Design*, 2005.

[75] A. V. S. S. Prasad, V. D. Agrawal, and M. V. Atre, "A New Algorithm for Global Fault Collapsing into Equivalence and Dominance Sets," *Proc. International Test Conf.*, Oct. 2002, pp. 391–397.

[76] W. A. Rogers and J. A. Abraham, "CHIEFS: A Concurrent, Hierarchical and Extensible Fault Simulator," *Proc. International Test Conf.*, March 2005.

[77] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronic Computers*, vol. EC-16, no. 5, pp. 567–580, Oct. 1967.

[78] R. K. K. R. Sandireddy and V. D. Agrawal, "Diagnostic and Detection Fault Collapsing for Multiple Output Circuits," *Proc. Design, Automation and Test in Europe Conf.*, March 2005.

[79] D. R. Schertz and G. Metze, "A New Representation for Faults in Combinational Digital Circuits," *IEEE Trans. Computers*, vol. C-21, no. 8, pp. 858–866, Aug. 1972.

[80] C. E. Stroud, "AUSIM: Auburn University SIMulator - Version 2.1," Technical report, Dept. of Electrical & Computer Engineering, Auburn University, March 12, 2004.

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

[82] R. R. Tummala and E. J. Rymaszewski, editors, *Microelectronics Packaging Handbook*. New York: Van Nostrand Reinhold, 1989. Page 677.

[83] A. Vaaje, "Theorems for Fault Collapsing in Combinational Circuits," *Journal of Electronic Testing: Theory and Application*. To be published.

[84] A. Veneris, R. Chang, M. S. Abadir, and M. Amiri, "Fault Equivalence and Diagnostic Test Generation Using ATPG," *Proc. IEEE International Symposium on Circuits and Systems*, May 2004, pp. 221–224.

[85] E. C. Weening and H. G. Kerkhoff, "A New Hierarchical Approach to Test-Pattern Generation," *Proc.* 4^{th} *IEEE International ASIC Conference and Exhibit*, Sept. 1991, pp. P6–1/1–4.

[86] H. C. Wittmann, B. H. Seiss, and K. J. Antreich, "Using Circuit Hierarchy for Fault Simulation in Combinational and Sequential Circuits," *Proc.* 4^{th} *European Conference on Design Automation*, Feb. 1993, pp. 432–436.

[87] P. Zhongliang, "Hierarchical Test Generation using Neural Networks for Digital Circuits," *Proc. International Conference on Neural Networks and Signal Processing*, Dec. 2003, pp. 245–248.

Appendices

Appendix A

A Sample Library File used in Hierarchical Fault Collapsing

This appendix describes a sample file containing collapsed set information as saved in the library of hierarchical collapsing technique. Such library files are used with the hierarchical fault collapsing program described in Appendix B.

The circuit M in Figure 5.2, whose collapsed set file is shown in this appendix, is repeated here as Figure A.1. Since this circuit has fewer gates than a pre-specified number, say 15, functional collapsing technique of the algorithm in Section 4.2.1 is used. The collapsed set file is as follows:

$INPUTs:

a 1 2

b 3 4*

$OUTPUTs:

g3 12 13

$TOTAL: 14

$FAULTs:

g1(0) 5

g1(1) 6

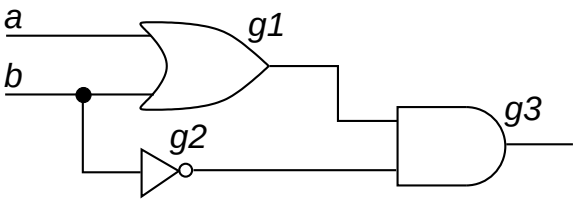

Figure A.1: Circuit M.

g2(1) 9

$RELATIONs:

1: 4 5 12 0

2: 6 13 0

3: 9 13 0

4: 1 5 12 0

5: 1 4 12 0

6: 2 13 0

9: 13 0

12: 1 4 5 0

13: 0

$REDUNDANT:

g1(b,0) 14

The entries under *$INPUTs:* and *$OUTPUTs:* correspond to the primary inputs and outputs of the circuit, respectively. The * on the input *b* indicates that *b* is a fanout stem in the circuit. This helps in deciding whether the line corresponding to this input has any significance at the level of hierarchy, where an instance of this circuit is used.

All faults are addressed by numbers. The two numbers against each input and output are the numbers representing stuck-at-0 and stuck-at-1 faults in that order. For example, line *a* stuck-at-0 is a fault numbered *1* and *a* stuck-at-1 is numbered *2*. *$TOTAL:* gives the total number of faults in the circuit. The entries listed under *$FAULTs:* are the faults of the collapsed set file that are neither on the input nor on the output line with the corresponding fault number. These faults are added to

the dominance matrix of any circuit, which uses an instance of the library circuit described in this appendix, irrespective of the way the library file is embed. Each line under *$RELATIONs:* lists the faults, ending with 0, that dominate the fault before colon (*:*). The redundant faults, if any, of the circuit are listed under *$REDUNDANT:*. For a circuit in which faults are structurally collapsed, the library file will not have any redundant faults.

Appendix B

Description of the Program used for Hierarchical Collapsing

This appendix explains the program used for collapsing the faults hierarchically. The program is written in the C language.

To start with, the input file (ISCAS'89 netlist format, often referred to as the bench format [21]) is read and the circuit structure is built. Corresponding to every primary input and gate in the circuit, there is a node in the linked list representing the circuit structure. The variables associated with every node are identified using the following data type:

```
struct list1
{    char name[20];
     int type;
     int fanouts;
     struct list5 *faults;
     struct list7 *fans;
     struct list7 *inputs;
     char visited;
}
```

The functionality of each variable in the structure is as follows:

name: name of the gate or primary input, which is assumed to be fewer than 20 characters.

type: type of the gate - an integer 1 indicates a primary input, 2 indicates a sub-circuit, 3 indicates an AND gate, and so on.

$fanouts$: number of fanouts to the gate.

$faults$: a pointer to the linked list which contains all stuck-at faults associated with the gate.

$fans$: a pointer to the linked list which contains pointers to the nodes representing its fanout gates.

inputs: a pointer to the linked list which contains pointers to the nodes representing its fanin gates.

visited: a flag used while performing equivalence collapsing.

First, the primary inputs are read through the *INPUT(name)* statements and a node is created. For every gate description in the netlist, a node is created and correspondingly a linked list pointed to by its *inputs* variable is created. For each node representing this gate's inputs, the linked list pointed to by $fans$ is updated with a pointer to the gate node. This is done for all the gates in the circuit description. A separate list of pointers is created to the nodes representing instances of sub-circuits.

Once the circuit structure is built, equivalence collapsed set is obtained using the function $findeq$. The function $findeq$ adds faults to the collapsed set, which is initially empty, based on the line collapsing technique described in Section 5.1. The function calls itself for each fanout node of the gate. This is a recursive function and terminates only when it reaches a primary output or a node that has already been visited. The variable *visited* is used for this purpose. The function is called once for each primary input ensuring that we visit all nodes in the circuit structure. Then the

dominance relations between the faults in the equivalence collapsed set are found as described in Section 5.1.1. The dominance matrix which represents the dominance relationships between faults, as described in Section 2.5, is implemented using a sparse matrix representation. Every fault is given a unique number to represent it in the sparse matrix.

Before dealing with sub-circuits, it is checked whether the outputs of all the instances of sub-circuits have both s-a-0 and s-a-1 faults in the collapsed set. If not, they are added so that there will be an equivalent fault already in the collapsed set. After inserting these fault relations, the transitive closure of the dominance matrix is computed using the *update* algorithm [33, 34]. Now, the library file corresponding to each sub-circuit used is read. If the library file is not found, it is created dynamically. If the number of gates in the sub-circuit is less than a pre-specified number, say 15, we use the ATPG-based functional collapsing technique as described in the algorithm of Section 4.2.1. Otherwise, the C program (that is, the one being described in this appendix) is called with the sub-circuit description as its input. Faults in the library file that are not on either input or output lines are added. The faults on the input and output lines are either merged with the corresponding lines at the higher level of hierarchy or neglected as described in Chapter 5. A sample library file is shown in Appendix A.

The *algorithm equivalence* and *algorithm dominance*, as explained in Section 3.2.1, are modified to suit the sparse matrix representation. *Algorithm Equivalence* removes all the extra faults that are added on the output lines of the instances of sub-circuits. Now, the library file containing the equivalent faults in the circuit is created for future

use in hierarchical collapsing. Just for the sake of library file, both s-a-0 and s-a-1 faults on the primary inputs are added, if not already present, and the corresponding dominance relationships with these faults are updated. Then the modified *algorithm dominance* is applied to obtain dominance collapsed set. Finally, the equivalence and dominance collapsing results are written onto an output file.